НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра автоматики та управління в технічних системах
(повна назва кафедри)

«На правах рукопису»
УДК _____

«До захисту допущено»
Завідуючий кафедрою
_____
_____
(підпис) (ініціали, прізвище)
“___”_____20__ р.

Магістерська дисертація

зі спеціальності (спеціалізації)_____
(код і назва спеціальності)

_____

на тему: Система управління якістю надання послуг критичними ІТ-інфраструктурами _____

_____

Виконав: студент __6__ курсу, групи ___ІА-82мп_____
(шифр групи)

Чанков Єгор Вадимович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник: зав. кафедрою АУТС, д.т.н., проф. Ролік О. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант: _ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент: _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) Інформатики та обчислювальної техніки
(повна назва)

Кафедра Автоматики та управління в технічних системах
(повна назва)

Освітньо-кваліфікаційний ступінь магістр
(назва ОКР)

Спеціальність 126 Інформаційні системи та технології
(код і назва)

Спеціалізація Інтегровані інформаційні системи
(назва)

ЗАТВЕРДЖУЮ
Завідуючий кафедрою
О.І. Ролік
(підпис) (ініціали, прізвище)
«_» _____ 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Чанкову Єгору Вадимовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації Система управління якістю надання послуг критичними ІТ-інфраструктурами
Науковий керівник дисертації зав. каф. АУТС, проф, д.т.н Ролік О.І.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «____» _____ 2019 р. № _____
2. Строк подання студентом дисертації _____
3. Об'єкт дослідження система управління якістю корпоративної ІТ-інфраструктури
_____
4. Зміст пояснювальної записки а) аналіз існуючих рішень; б) розробка структурної схеми системи; в) розробка системи управління якістю надання послуг; г) розробка алгоритму управління; д) розробка стартап-проекту.
5. Перелік завдань, які потрібно розробити: проаналізувати поняття якості послуг ІТ-інфраструктур, проаналізувати існуючі рішення систем управління якістю надання послуг критичними ІТ-інфраструктурами, розробити модель управління якістю, розробити структурну схему системи, протестувати розроблену систему, провести маркетинговий аналіз стартап проекту.
6. Перелік графічного (ілюстративного) матеріалу: структурна схема системи управління якістю надання послуг критичними ІТ-інфраструктурами, схема алгоритмів роботи та тренування моделей машинного навчання, схема класів, схема

розгортання компонентів, схема бази даних, діаграма прецедентів, діаграма послідовностей.

7. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

8. Дата видачі завдання 04 вересня 2019

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	Аналіз предметної області, огляд літератури	15.09.2019	
2	Дослідження методів управління якістю послуг	18.09.2019	
3	Збір даних та розробка моделі машинного навчання	30.09.2019	
4	Технічне рішення та реалізація системи управління якістю	20.10.2019	
5	Тестування та аналіз роботи системи	10.11.2019	
6	Розробка стартап-проекту	20.11.2019	
7	Оформлення матеріалів дисертації	04.12.2019	

Студент _____ Є. В. Чанков
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ О.І. Ролік
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Магістерська дисертація 128 с., 25 рис., 22 табл., 8 дод., 18 дж.

Тема магістерської дисертації: «Система управління якістю надання послуг критичними IT-інфраструктурами».

Актуальність роботи полягає в тому, що якби проблеми в критичних IT-інфраструктурах можна було б виявляти заздалегідь, можна було б вчасно застосувати відповідні дії для уникнення таких ситуацій. Використання системи стане одним з факторів збільшення конкурентоспроможності підприємства та дозволить економити значні ресурси в майбутньому.

Об’єктом дослідження є система управління якістю надання послуг критичною IT-інфраструктурою.

Предметом дослідження є показники якості надання послуг критичною IT-інфраструктурою.

Дана робота має на меті створення системи управління якістю надання послуг критичними IT-інфраструктурами, основною задачею якої є предиктивний моніторинг стану системи, оповіщення про можливі несправності у майбутньому на основі аналізу попередніх даних про використання ресурсів системою.

Для досягнення такої цілі необхідно:

1) визначити, які метрики впливають на стан системи;
2) проаналізувати сильні та слабкі сторони аналогів;
3) провести огляд існуючих рішень;
4) провести огляд необхідних технологій;
5) реалізувати модуль предиктивного аналізу;
6) протестувати розроблене програмне забезпечення.

КЛЮЧОВІ СЛОВА: ПРЕДИКТИВНИЙ АНАЛІЗ, ШТУЧНІ НЕЙРОННІ МЕРЕЖІ, УПРАВЛІННЯ ЯКІСТЮ, КРИТИЧНА IT-ІНФРАСТРУКТУРА.

ABSTRACT

Master's Thesis 128 p., 25 figures, 22 tables, 8 appendixes, 18 sources.

Theme of the master's thesis is "Quality management system for critical IT infrastructure service delivery".

The reason for the relevance of a master's thesis is if the problems that arise in critical IT-infrastructure were detected in advance, it could be possible to fix them and avoid such situations in the future. The use of such system may lead to rise of the enterprise competitiveness and resource economy.

The object of the study is the quality management system for critical IT infrastructure service delivery.

The subject of the study is the metrics of quality of critical IT infrastructure service delivery.

The purpose of this work is to create the quality management system for critical IT infrastructure service delivery. The major task of such system is a predictive monitoring of system’s state, alerting in the situation of possible critical failures in the future based on the resource usage historical data.

To achieve this goal, it is necessary to:

1) determine, which metrics affects the state of the system;
2) analyze the strengths and weaknesses of analogues;
3) review existing solutions;
4) review the necessary technologies;
5) implement a predictive analysis module;
6) test the developed software.

KEYWORDS: PREDICTIVE ANALYSIS, ARTIFICIAL NEURAL NETWORKS, QUALITY MANAGEMENT, CORPORATE IT-INFRASTRUCTURE.

Пояснювальна записка

до магістерської дисертації

на тему: «Система управління якістю надання послуг критичними IT-інфраструктурами»

Київ – 2019 рік

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

APM – Application performance management

AWS – Amazon web services

CPU – Central processing unit

CSV – Comma separated values

DevOps – Development and operations

ELK – Elastic Logstash Kibana

HPA – Horizontal Pod Autoscaler

HTTP – Hypertext transfer protocol

IDE – Integrated development environment

ITIL – Information technology infrastructure library

ITSM – Information technology service management

JSON – JavaScript object notation

KQL – Kibana query language

OSI – Open Systems Interconnection

REST – Representational state transfer

SLA – Service level agreement

SLM – Service level management

SPL – Splunk library

SQL – Structured query language

UML – Unified modelling language

WPA – Wider Pod Autoscaler

БД – База даних

ІТ – Інформаційні технології

ООП – Об’єктно-орієнтоване програмування

СУБД – Система управління базами даних

ЦОД – Центр обробки даних

ВСТУП

Для сучасного підприємства наявність розвиненої ІТ-інфраструктури є ключовим фактором її конкурентоспроможності [1]. Завдяки комп'ютеризованим системам управління компанії не лише збільшують свою продуктивність, адже в порівнянні з людиною, комп'ютер може оброблювати одночасно великі масиви даних та приймати на основі них рішення з вели-кою швидкістю, а ще й зменшують витрати на людські ресурси, виключається так званий людський фактор.

Проте існують підприємства, на яких навіть найменша помилка в ІТ-інфраструктурі може призвести до катастрофічних наслідків для країни. ІТ-інфраструктури таких підприємств називають критичними. Разом з назвою до них ставлять більш високі вимоги щодо швидкодії, відмовостійкості, кібербезпеки тощо. Прикладом таких критичних ІТ-інфраструктур можуть слугувати ІТ-інфраструктури електростанцій, заводів, транспортної системи та парламентів.

У 2003 році відмова критичної ІТ-інфраструктури, що підтримувала роботу компанії, яка керувала електронними мережами на північному сході Америки, призвела до того, що протягом 14 днів більш, ніж 50 мільйонів людей жили без світла [2]. Всі критичні об'єкти інфраструктури також залишилися знеструмленими. Це стало одним з найбільших знеструмлень в історії та нанесло збитків на сотні мільйонів доларів.

Найбільше знеструмлення в історії сталося в 2012 році на півночі Індії [3], яке призвело до того, що протягом двох днів близько 620 мільйонів жителів країни, тобто близько 9% від населення світу на той момент, залишилися без електроенергії. Знеструмленими залишилися не лише будинки людей, а ще й заводи – зупинилося виробництво, ферми, банківські установи, лікарні та служби швидкого реагування. Збитки від такого знеструмлення оцінювалися в декілька мільярдів доларів, а на те, щоб відновити 80% роботи інфраструктури пішло 15 годин.

За законом Законом України Про кібербезпеку поняття критичності також поширюється на будь-які ІТ-інфраструктури, які мають важливе значення для підтримки життєво-важливих соціальних функцій навіть в менших масштабах. В

даній роботі у якості критичної IT-інфраструктури було використано інфраструктуру відкритих державних онлайн-аукціонів ProZorro, яка має підвищені стандарти безпеки та відмовостійкості для забезпечення безперебійної роботи десятків майданчиків онлайн-аукціонів і сотень аукціонів, які відбуваються щодня.

Кожного року компаніями витрачаються більше 4% від свого прибутку на підтримання IT-інфраструктури, що в загальному випадку становить близько 6,4 тис. доларів на людину на рік. Із року в рік ця цифра збільшується, адже зростає комп'ютеризованість підприємств, постійно оновлюється апаратне забезпечення, розширюється функціонал на-явного програмного забезпечення тощо. Найбільшу статтю розходів складає адміністрування та підтримка корпоративної інфраструктури, а четверту за величиною – розходи на ресурси серверів.

До процесів адміністрування та підтримки IT-інфраструктур можна віднести, наприклад, налаштування мережі, встановлення програмного забезпечення, створення конфігурацій а також, якщо говорити про віртуальні рішення, то це виділення ресурсів. Так при створення нового віртуального середовища адміністратор повинен визначити, скільки ресурсів сервера, таких як кількість віртуальних ядер процесора, кількість процесорів або процесорного часу, оперативної пам'яті та місця на диску потрібно виділити. Від цього вибору залежить швидкодія сервісів, а неоптимальний вибір матиме наслідки у вигляді простою ресурсів та призведе до матеріальних збитків у майбутньому.

При цьому адміністратор відіграє ключову роль в підтримці належно-го рівня функціонування критичної IT-інфраструктури та безперебійності роботи наявних вузлів системи. Якщо віртуальних ресурсів буде недостатньо, це спричинить погіршення продуктивності роботи компонентів, що недопустимо для критичної IT-інфраструктури. З іншого боку, якщо ресурсів буде виділено необдумано багато, то більшість часу ресурси будуть простоювати та не використовуватися. В такому разі в рамках одного набору ресурсів можливо буде створити набагато менше віртуальних середовищ. В обох випадках це призведе до збільшення витрат: в першому – через

погіршення продуктивності та збільшення часу обробки даних, а в другому – через необхідність придбання додаткових ресурсів.

Актуальність розробки підсистеми управління якістю надання послуг критичними IT-інфраструктурами підтверджується тим, що за статистикою більшості проблем, які виникають в критичній IT-інфраструктурі, можна було б уникнути, якби стандарти захисту систем від неполадок були набагато вищими. Якби проблеми в критичних IT-інфраструктурах можна було б виявляти заздалегідь, можна було б вчасно застосувати резервні ресурси, наявні в системі. На жаль, після того, як проблема стається, вже занадто пізно шукати проблему та намагатися її вирішити, адже збитки нанесені корпорації або в найгіршому випадку країні вже на третю хвилину після виходу з ладу можуть оцінюватися мільйонами доларів. Розробка такої системи стане одним з факторів збільшення конкурентоспроможності підприємства та дозволить економити значні ресурси в майбутньому.

Дана робота має на меті створення системи управління якістю надання послуг критичними IT-інфраструктурами, основною задачею якої є предиктивний моніторинг стану системи, оповіщення про можливі несправності у майбутньому на основі аналізу попередніх даних про використання ресурсів системою. Об’єктом дослідження є система управління якістю надання послуг критичною IT-інфраструктурою, а предметом дослідження є показники якості надання послуг критичною IT-інфраструктурою.

Для досягнення такої цілі необхідно:

1) визначити, які метрики впливають на стан системи та які метрики використовуються для аналізу стану системи;

2) проаналізувати сильні та слабкі сторони різних систем для збору та аналізу логів та метрик;

3) провести огляд існуючих рішень до управління якістю надання послуг критичними IT-інфраструктурами;

4) провести огляд необхідних технологій та проаналізувати аналогічні рішення, щоб визначитися з подальшими діями щодо програмної реалізації системи;

5) реалізувати модуль, який на основі предиктивного аналізу історичних відомостей про використання ресурсів в системі та можливих помилок, пов’язаних з цим, зможе приймати рішення щодо управління системними ресурсами;

6) провести тестування розробленого програмного забезпечення та підтвердити працездатність моделі експериментальним шляхом.

У ході роботи для досягнення поставленої мети було використано наступні методи: алгоритм машинного навчання рекурентна нейронна мережа, перехресна валідація моделі, генерація даних, поділ даних на тестову та тренувальну вибірки, прототипування системи за допомогою UML-діаграм.

Бакалаврський проект складається з наступних розділів: вступ, п’ять основних розділів, висновки, список використаних джерел із 18 найменувань. Графічна частина включає в себе 8 креслеників формату А3. Загальний обсяг сторінок – 141.

1 АНАЛІЗ КОНЦЕПЦІЇ УПРАВЛІННЯ ЯКІСТЮ НАДАННЯ ПОСЛУГ ІТ-ІНФРАСТРУКТУРАМИ

1.1 Постановка проблеми управління якістю надання послуг критичними ІТ-інфраструктурами

Управління рівнем обслуговування або SLM визначається як «відповідальність за забезпечення того, щоб усі процеси управління сервісом, угоди про рівень надання послуг та контракти, що підкріплюються, відповідали узгодженим цілям рівня обслуговування. SLM включає в себе відстежування та звітування про рівні обслуговування та регулярні огляди зі сторони клієнтів.» Іншими словами, ключовими критеріями будь-якої інформації, яка повинна міститися в Угоді про рівень обслуговування (SLA), є те, що це повинна бути вимірювана величина, оскільки угода має бути чіткою для розуміння та стислою [4]. Угода про рівень надання послуг – це договір між постачальником послуг та його внутрішніми або зовнішніми замовниками, який документує, які послуги надаватиме постачальник, та визначає стандарти послуг, якими зобов'язаний відповідати постачальник.

Протягом багатьох років угоди про рівень надання послуг все частіше використовувалися для регулювання зростаючого набору моделей закупівель в галузі ІТ. Коли аутсорсинг в галузі ІТ з'явився наприкінці 1980-х років, угоди про рівень надання послуг розвивалися як механізм управління такими відносинами між компанією-замовником та компанією-виконавцем. Такі угоди встановлюють очікування щодо ефективності постачальника послуг та встановлюють штрафні санкції за пропущення цілей, а в деяких випадках – і бонуси за їх перевищення. Оскільки проекти на аутсорсингу часто підганялися під конкретного замовника, аутсорсингові угоди про рівень надання послуг часто розроблялися для регулювання конкретного проекту.

1.1.1 Використання угод про рівень надання послуг IT-інфраструктурами

По мірі того, як керовані сервіси та послуги хмарних обчислень стали більш поширеними в останні роки, угоди про рівень надання послуг еволюціонували для регулювання цих підходів. Використання спільних ресурсів, а не окремо виділених, як це було раніше, потребувало новіших методів укладання контрактів, тому такі угоди, як правило, включають в себе широкий спектр понять та призначаються для покриття всіх потреб клієнтів постачальника послуг. SLA, незалежно від типу, можуть змінюватися з часом [5]. Постачальники послуг періодично перевірятимуть та оновлюватимуть угоди про рівень обслуговування, щоб відобразити появу нових послуг, зміни до існуючих послуг або зміни у загальному нормативному середовищі.

Постачальникам послуг така угода про рівень надання послуг потрібна в першу чергу для того, щоб постачальники мали змогу керувати очікуваннями клієнтів та визначати обставини, за яких вони не несуть відповідальності за перебої або проблеми з роботою. Клієнти також можуть отримати вигоду від угод про рівень надання послуг, оскільки вони описують експлуатаційні характеристики послуги, які можна порівняти з угодами про рівень надання послуг інших постачальників, а також визначити засоби для вирішення питань щодо обслуговування.

Що стосується постачальника послуг, то угода про рівень надання послуг, як правило, є однією з двох основних угод, які постачальник заключає з клієнтами. Багато постачальників послуг заключають одну основну угоду про рівень надання послуг, щоб встановити загальні умови, в яких вони працюватимуть із клієнтами. SLA часто включається в договори шляхом посилання на нього в основному договорі постачальника послуг. Між двома угодами про надання послуг, Угода про рівень надання послуг надає більшої специфіки щодо наданих послуг та показників, які будуть використані для вимірювання їх ефективності.

У широкому розумінні, угода про рівень надання послуг зазвичай включає виклад цілей, перелік послуг, які підпадають під дію угоди, а також визначає обов'язки постачальника послуг та замовника відповідно до угоди. Наприклад, замовник несе відповідальність за надання представника для вирішення питань з

постачальником послуг у зв'язку із угодою про рівень надання послуг. Постачальник послуг буде нести відповідальність за дотримання рівня обслуговування, визначеного угодою.

Угода про рівень надання послуг передбачає також розділ, в якому детально описуються виключення, тобто ситуації, в яких гарантії угоди, а також штрафи за їх невиконання не застосовуються. У цьому списку можуть бути такі події, як стихійні лиха чи терористичні акти. Цей розділ іноді називають загрозою форс-мажорних обставин, яка має на меті позбавити постачальника послуг відповідальності за збитки від подій, що не підлягають його контролю.

Вважається, що угоди про рівень надання послуг SLA виникли у постачальників мережевих послуг [6, 7], але зараз широко використовуються в ряді IT-галузей. Компанії, що встановлюють угоди про рівень надання послуг, включають постачальників послуг IT та постачальників послуг хмарних обчислень. Корпоративні IT-організації, зокрема ті, які прийняли управління IT-послугами (ITSM), заключають угоди про рівень надання послуг з власними клієнтами – користувачами в інших підрозділах підприємства. IT-відділ створює угоду про рівень надання послуг, щоб її послуги можна було оцінити, виправдати і, можливо, порівняти з послугами аутсорсингових постачальників.

1.1.2 Використання методів прогнозування для дотримання узгодженого рівня надання послуг IT-інфраструктурами

Коли внутрішні IT-підрозділи приймають угоду про узгодження рівня надання послуг, штраф за збій, як правило, є лише тратою часу. Немає сенсу звільняти працівників, які керують усім центром обробки даних через те, що додаток повільно обробляє запити. Підприємства можуть передавати додаток для встановлення на сервери постачальників послуг, але проблема вимірювання метрик угоди про рівень надання послуг залишається.

При чому варто зазначити, що оцінку якості надання послуг IT-інфраструктурами експерти надають зазвичай однобічно, не розглядаючи картини в

цілому. Так, наприклад, на якість надання ІТ-інфраструктурою послуг можуть мати вплив не лише події та процеси в самій системі вже після її створення та впровадження, а також сам процес проектування та розробки системи. І це цілком закономірно, адже якість впровадження системи має вплив на всю систему в подальшому. Тому дуже важливо, щоб компанія довіряла свою безпеку спеціалістам, які мають експертизу в галузі.

Більшість підприємств мають мережу поєднаних серверів, сховищ, мереж та застосунків як сервіс від своїх партнерів, що надають такі послуги, що, і кожен з них вимагає узгодження через заключення окремої угоди про рівень надання послуг [8]. Угода про рівень надання послуг повинна забезпечити досягнення потрібного рівню обслуговування в першу чергу. Угода повинна висвітлювати, як система працювали в минулому, як вона працює зараз і як вона повинна працювати в майбутньому.

Прогнозний моніторинг є ключовим фактором успішного дотримання угоди про рівень надання послуг. Наприклад, один з показників SLA визначає прийнятну швидкість реакції на скарги кінцевих користувачів. У документі в цьому прикладі зазначено, що середнє значення, виміряне протягом години в 500 мілісекунд, є прийнятним. Коли угода починає діяти, це середнє значення дорівнює 300 мс, що нижче встановленого погодинного періоду угоди [9]. Однак час реакції повільно збільшується з часом, оскільки бізнес продовжує використовувати додаток: 350 мс, 405 мс, потім 470 мс. Якщо стежити лише за значенням одного з параметрів, то такі операції все ще знаходяться в межах прийнятних параметрів. Але якщо спробувати спрогнозувати стан системи в майбутньому, можна заздалегідь побачити, що час реакції стане проблемою, перш ніж угода про рівень надання послуг буде порушена.

Тому для того, аби підтримувати рівень надання послуг на узгодженому рівні, потрібно слідкувати за тенденціями та вживати заходів до того, як буде порушена угода про рівень надання послуг. Найкраще рішення такої задачі – налаштування системи, що в довгостроковій перспективі приведе до покращення значень метрик, наприклад це може означати оптимізація коду чи масштабування обладнання. Варто також визначити причину проблеми. Чи сповільнюється час реакції через проблеми

із додатком або через збільшення кількості користувачів? Якщо програма неефективна, можна визначити першопричину, наприклад, витік пам'яті, та спробувати оптимізувати на рівні програми. Якщо проблема пов’язана з додатковими користувачами, потрібно масштабувати ресурси центру обробки даних для даного додатку.

Через постійне зростання використання цифрових технологій та мінливим навантаженням постачальник послуг повинен проконсультувати бізнес щодо того, що потрібно для того, щоб залишатися в межах рівня обслуговування, або якщо очікуваний рівень послуг нереально досягнути. Наприклад, якщо перенасичення користувачів сповільнює додаток, більша віртуальна машина може усунути проблему, але це коштуватиме більше грошей. Якщо бюджет не дозволить отримати більше ресурсів, можна домовитися про збільшення вартості надання послуг.

Переговори щодо дотримання внутрішньої угоди про рівень надання послуг є більш складною задачею. Неоднорідна структура різних центрів обробки даних, що перебувають у власності компанії, що надає послуги, у поєднанні з колокаційними та хмарними послугами означають, що "головна" угода про рівень надання послуг повинна підходити і внутрішньому IT-відділу компанії, і зовнішнім постачальниками послуг.

Угоди про рівень надання послуг повинні бути угодами, що діють на постійній основі, а не документами, які забуваються. Компанія повинна регулярно переглядати такі угоди, щоб переконатися, що існуючі показники все ще відповідають їхньому призначенню. Бізнес повинен брати участь у цих оглядах – час відгуку в 500 мс, погоджений три місяці тому, може бути занадто повільним сьогодні, оскільки конкурент вийшов на ринок. Завдяки перегляду угод про рівень надання послуг, бізнес дізнається, які інвестиції можуть знизити цільовий час реакції до 350 мс або 400 мс, замість того, щоб просто потребувати необґрунтованого відшкодування штрафів, зазначених у договорі про рівень надання послуг. Саме тому обговорювати штрафні санкції краще за все лише тоді, коли угода про рівень надання послуг була дійсно порушена.

1.1.3 Аналіз критичної IT-інфраструктура відкритих онлайн аукціонів

Система електронних закупівель є новою для України, тому цілком закономірно виникають питання: що таке Прозоро, як працювати з ним, в чому полягає відмінність від старих паперових державних тендерів. З запровадженням відкритих державних аукціонів, торги проходять в режимі онлайн. Оголошення про покупку розміщується державним замовником і містить такий перелік:

- предмет закупівлі;
- ціна;
- вимоги.

Підприємець дізнається про початок торгів безпосередньо на сайті або за допомогою інформаційної розсилки на мобільний телефон або email. У тому випадку, якщо оголошення зацікавило, учаснику потрібно увійти в особистий кабінет і подати свою пропозицію. Після закінчення аукціону система розкриває ціни, залишаючи заявників анонімними. Що таке програма ProZorro і як з нею працювати, зрозуміти досить легко, оскільки це зворотній аукціон. Тобто, аукціон на пониження ціни – редукціон.

Аукціон відбувається в 3 кроки. Кожен з учасників в процесі аукціону може знизити свою початкову пропозицію. Система визначає учасника, який пропонує найнижчу ціну і відкриває інформацію по всіх учасниках після завершення торгів. Підприємець, чия пропозиція була визначена низькою, має пройти кваліфікацію і підтвердити документально, що його товар або роботи або послуги відповідають вимогам, які висунув замовник у тендерній документації.

Визначення найнижчої пропозиції відбувається за умовами, які замовник позначив в оголошенні про проведення аукціону мінімальний крок зниження ціни, спосіб визначення мінімального кроку – це може бути процентний відсоток, грошові одиниці або математична формула. Програма побудована таким чином, що учасник з найнижчою ставкою має можливість зробити останню пропозицію в раунді. Він може коригувати свою ставку в залежності від ставок конкурентів.

Система регулюється відповідно до Закону України «Про державні закупівлі» від 25.12.2015 року під номером 922-VIII. Використання державними установами ProZorro не вимагає від них витрат, а навпаки – дозволяє значно економити на закупівлі, оскільки учасники торгів зацікавлені запропонувати самі низькі ціни.

Система ProZorro – це розташована на серверах база даних, до якої не можна підключитися безпосередньо, а тільки через одну з торгових майданчиків, які мають до неї доступ. В даний час до системи приєднані десятки майданчиків, а кожного місяця готуються до запуску ще декілька. Завдяки цьому держава або нинішній адміністратор бази – українське представництво міжнародної організації Transparency International – не можуть впливати на учасників торгів і будь-яким чином штучно відсівати їх. Майданчики самі зацікавлені в тому, щоб знайти якомога більше постачальників.

Через те, що навантаження на систему в пікові моменти може сягати сотень тисяч запитів в хвилину, і через очевидну важливість такого роду майданчиків, дуже важливо забезпечувати відповідний рівень надання послуг такими IT-інфраструктурами. Для того, щоб зрозуміти проблему в цілому, потрібно проаналізувати окремі складові критичної IT-інфраструктури.

1.2 Огляд інструментів аналізу якості надання послуг критичними IT-інфраструктурами

В процесі розробки програмного продукту будь-який розробник стикається з помилками, викликаними неправильною послідовністю виконання певних дій, алгоритмами, що були розроблені неправильно, або якщо при їх проектуванні не було враховано всіх можливих комбінацій вхідних даних [10]. В кращому разі виконання такої програми призведе до некоректної її роботи та екстреного закриття цього додатку. Таке часто відбувається при написанні програм некваліфікованими спеціалістами, які не використовують додаткові інструменти для роботи з помилками,

такі як перехоплення та обробка помилок в коді, перехоплення та обробка помилок на рівні модулю або програми, точки зупинки, налагодження, логування тощо.

В найпростішому вигляді процес налагодження програми виглядає, як зупинка виконання програмою роботи на деякий час, що дає змогу програмісту перевірити значення локальних змінних, змінити їх за необхідністю і навіть динамічно виконати частину коду. Коли людина закінчує досліджувати стан програми в одному місці коду, можна продовжити виконання програми аж до наступної зупинки. Таку можливість зазвичай надають інтегровані середовища розробки (IDE). Такі програмні продукти, маючи інтуїтивно зрозумілий інтерфейс та можливість використовувати гарячі клавіші, значно пришвидшують процес налагодження програм, розроблених користувачами, та, відповідно, допомагають швидше виявити помилку, полагодити проблемне місце та протестувати роботу заново.

Проте розробка простих програмних продуктів, таких як прикладні програмні інтерфейси, додатки, що виконуються в одному потоці, бекенд для веб-сайтів, бібліотеки, додатки для десктопів тощо, сильно відрізняється від розробки більш складних додатків, в яких багато складових частин працюють окремо одна від одної. До таких програм можна віднести програми, що виконуються в декількох потоках паралельно, високонавантажені програми, що паралельно оброблюють мільярди байтів інформації тощо. В першому випадку достатньо запустити програму в режимі налагодження та перевірити правильність її виконання крок за кроком, що відбувається доволі швидко та з мінімальними затратами часу розробника для налаштування. У другому ж випадку, неможливо налагодити такі програми стандартними інструментами з багатьох причин: непослідовність виконання коду, паралельне виконання декількох можливих підпрограм, результат яких цікавить, залежність результату однієї підпрограми від результату іншої.

В таких випадках постає питання, як же налагоджувати таке програмне забезпечення? Тут на допомогу приходить поняття «логування». З точки зору програмного продукту логування виглядає як декілька додаткових строчок коду, які мають на меті виводити в зовнішній буфер певні дані з середини програми. Зазвичай

виводять значення внутрішніх змінних в програмі на момент логування, а також повідомлення про початок, кінець певної логічної послідовності дій і, що дуже важливо, можливі помилки. Наприклад, типова обробка помилок складається зі спроби виконати код в середині спеціальної синтаксичної конструкції, блок перехоплення помилок, в якому розташоване логування типу помилки, повідомлення та додаткових значень, і блок, який виконується вкінці-кінців, не залежно від того, чи відбувалася помилка чи ні. Тому логування є невід'ємною частиною процесу пошуку причин та усунення помилок.

Варто зазначити, що частини програм, які відповідають за логування, зазвичай мають гнучкі налаштування, тобто можна обирати не лише такі параметри, як рівні логування, вихідний буфер – на консоль, в інший додаток, на файлову систему, в потік пам'яті, – формат повідомлення, формат дати і часу тощо. Інтерфейси такі підпрограми мають стандартизовані для того, щоб користувачеві не потрібно було годинами читати документацію, а було достатньо скачати приклад коду та інтегрувати його в своє рішення.

1.2.1 Аналіз проблеми логування

В сучасному світі з розвитком веб технологій кожна секунда очікування клієнта зменшує його бажання користуватися додатком в десятки разів, секунда затримки на серверах, які займаються біржовою торгівлею, коштують мільйонів доларів, а секунда простою серверів критичної IT-інфраструктури може навіть коштувати людських життів. Проблеми з продуктивністю можуть нашкодити компанії [11], але разом з тим компанії також не можуть дозволити собі втрати грошей та клієнтів через такі проблеми, а також штрафи та санкції і судові процеси в гіршому випадку.

Щоби впевнюватися в тому, що інфраструктура працює та забезпечує певний рівень послуг, інженери покладаються на різні типи даних, згенерованих програмними продуктами та IT-інфраструктурою, яка їх підтримує. Такі дані – логування подій або метрики, або й те й інше – надають змогу відстежувати стан таких систем та швидко знаходити і виправляти помилки, якщо вони з'являються.

Але логування як таке існувало завжди, тому й за весь час існування комп'ютерних систем, розвивалися інструменти для їх аналізу. Те, що змінилося з часом, це архітектура таких систем, які генерують логи [12]. Звичайні додатки еволюціонували в мікросервісні серверні архітектури, використання технологій контейнеризації та оркестрації при побудові IT-інфраструктур, які тепер все частіше використовують хмарні рішення або гібридні застосування. Тому частка даних, які генерують ці системи не просто стрімко зростає, вони тепер представляють для своїх розробників цілу проблему. Проблемою стало навіть звичайне читання файлу з логами, адже розміри таких файлів на сервері може досягати терабайтів протягом одного дня [13]. Загалом, будь-яке сучасне рішення з управління логами повинне включати в себе наступні ключові якості:

- агрегація логів – це можливість збирати логи з багатьох джерел даних одночасно та пересилання їх до сховища;
- обробка логів – можливість системи трансформувати та виокремлювати з повідомлень логів змістовні дані;
- зберігання логів – можливість зберігати дані протягом довгого відрізку часу для надання можливості відслідковування, аналізування трендів та покращення безпеки;
- аналіз логів – можливість надсилати запити для створення певної вибірки даних для створення візуалізації та інформаційних панелей.

Саме тут варто згадати про централізовані рішення зі збору та зберігання даних, наприклад як ELK stack або Splunk.

1.2.2 Аналіз програмного забезпечення Splunk

Програмне забезпечення Splunk досить довгий час вважалося лідером в сфері спостереження за реальним рівнем надання послуг підприємствами. Дане програмне забезпечення стало піонером в сфері моніторингу за станом системи в реальному часі, сповіщення та усунення несправностей, використовуючи метрики і логи – двох із трьох основних компонентів будь-якої системи спостереження за станом системи. А нещодавно розробники цього програмного забезпечення також розширили

можливості спостереження за допомогою інтеграції з FireLens, новою службою агрегації логів від AWS.

Програмне забезпечення Splunk являє собою користувацький інтерфейс для обробки даних, а також механізм збереження даних в базу даних і механізм агрегації логів. Хоча через користувацький інтерфейс можна виконувати пошук з використанням простих пошукових термінів, наприклад ім’я користувача та побачити, як часто це з’являється за певний часовий період, пошукова технологія обробки Splunk (SPL) пропонує набагато більше. SPL – це надзвичайно потужний інструмент для фільтрації величезної кількості даних та виконання статистичних операцій щодо того, що є актуальним у конкретному контексті. Наприклад, за допомогою Splunk можна дізнатися, які програми найповільніше запускалися, змушуючи користувача чекати найдовше.

Варто додати, що сам по собі додаток Splunk нічого не знає про дані, але він може отримувати дані з найрізноманітніших джерел: різноманітні види логів, журнали подій Windows, Syslog тощо. Якщо потрібні дані не можуть бути знайдені в жодному журналі логів, можна написати власний сценарій та налаштувати Splunk, таким чином, щоб на вихід можна було отримувати оброблені дані в потрібному вигляді. Плагіни для додатку Splunk можуть бути використані як для обробки вхідних даних, так і для розширення інформаційної панелі.

Програму Splunk часто асоціюють з базами даних, але це помилкова думка. Якщо база даних вимагає, щоб користувач визначив таблиці та поля, перш ніж зберігати дані, Splunk приймає дані майже будь-якого вигляду одразу після встановлення. Іншими словами, у Splunk немає фіксованої схеми. Натомість цей додаток виконує пошук поля автоматично під час роботи. Багато форматів логів розпізнаються автоматично, все інше можна вказати у файлах конфігурації або прямо в виразі пошуку. Такий підхід забезпечує велику гнучкість. Додаткові налаштування дають змоги на основі аналізу даних створювати системні оповіщення. Під час фази індексації, коли Splunk обробляє вхідні дані та готує їх для зберігання, індексатор вносить одну істотну модифікацію: він розбиває потік символів на окремі події.

Події зазвичай відповідають рядкам у файлі логів, який обробляється. Кожна подія отримує часову позначку, як правило, проаналізовану безпосередньо з рядка введення, та кілька інших властивостей за замовчуванням, таких як машина, з якої були надіслані логи. Потім ключові слова події додаються в індексний файл для прискорення подальших пошуків, а текст події зберігається у стисненому файлі, що знаходиться прямо у файловій системі. Тобто істотною особливістю цього програмного забезпечення є відсутність резервного сервера для управління даними, а також відсутність бази даних, до якої можна приєднатися зі сторонніх додатків. Splunk зберігає дані безпосередньо у файловій системі. І це має такі переваги:

- установка надзвичайно швидка. Splunk доступний для більшості сучасних операційних систем, а установка зазвичай займає декілька хвилин;
- проста масштабованість. Якщо одного сервера Splunk недостатньо, можна просто додати ще один. Дані, що надходять, автоматично розподіляються рівномірно по всім серверами, а пошуки спрямовуються до всіх екземплярів програми Splunk, завдяки чому швидкість зростає зі збільшенням кількості машин;
- немає жодної точки відмови. Адже в системі немає перевантаженого сервера баз даних, який часто є причиною уповільнення роботи програм у центрах обробки даних;
- нескінченне зберігання даних. Деякі програми моніторингу дозволяють зберігати стільки даних, скільки місяців, тижнів чи навіть днів пройшло від моменту події. Splunk може індексувати сотні терабайт даних на день і зберігати практично необмежену кількість даних, не стискаючи старі дані.

Проте недоліком Splunk можна назвати те, що програма обмежує кількість нових даних, які можна індексувати за день при використанні безкоштовної підписки. Доступна безкоштовна версія, що має обмеження 500 мегабайтів даних на день. Існують платні ліцензії для програми, такі як Splunk Enterprise, які надають можливість індексувати гігабайти даних в день. Кількість серверів Splunk, на яких зберігаються дані, залежить від того, як довго зберігаються дані або протягом якого періоду часу здійснюється пошук.

1.2.3 Аналіз програмного забезпечення ELK stack

Як ще один приклад, розглянемо програмне забезпечення ELK stack. Назва це абревіатура, що походить від перших букв назв трьох основних програмних компонентів, які складають дану технологію: Elasticsearch, Logstash та Kibana. За декілька років, що дана технологія була презентована на ринку, ELK скачали та встановили вже більше декількох мільйонів разів. Таким чином ELK стала найпопулярнішою платформою для управління логами, обігнавши своїх найближчих конкурентів багатократно.

Власне платформа розвивалася з декількох окремих програмних продуктів, що розроблялися і підтримувалися компанією Elastic не один рік. Але врешті-решт із доданням важливого компоненту – програми Beats – система сформувалася як самодостатня екосистема та змінила назву на ELK stack, як показано на рисунку 1.1. Популярність екосистеми пояснюється тим, що до цього на ринку не було випущено жодної повноцінної системи, здатної обробляти мільйони байтів інформації та масштабуватися, щоб задовольняти потреби корпоративних IT-інфраструктур різних масштабів.

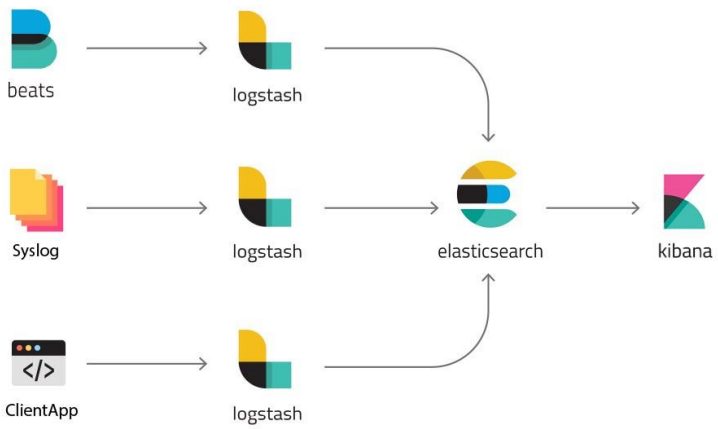

Рисунок 1.1 – Схематичне зображення компонентів ELK stack

Також на відміну від інших рішень, екосистема ELK stack складається з компонентів, що мають виключно відкритий код. Використання таких продуктів

допомагає підприємствам не лише економити гроші за використання продуктів, платних в іншому разі, а й запобігають монополізації використання продуктів лише одним виробником. До того можна додати також той факт, що знаходити нових членів команди, які розбираються в програмному забезпеченні з відкритим вихідним кодом набагато простіше.

Оскільки екосистема складається з чотирьох основних компонентів, варто розглянути їх окремо. Elasticsearch це ядро всієї системи, якому відведено ключову роль у підтримці правильної та безперебійної роботи кожної з компонент. Цей компонент хоч і використовується лише задля пошуку та аналізу логів, проте цього більш ніж достатньо для того, щоб комп'ютерна програма Elasticsearch стала однією з найпопулярніших рушіїв для баз даних сьогодення.

1.2.3.1 Огляд програмного забезпечення Elasticsearch

Вперше програму презентували ще у 2010 році. Популярність системі приніс той факт, що вона має відкритий вихідний код, написаний на одній з найпрогресивніших мов програмування Java. Мова Java це проект, який почався ще в кінці дев'яностих років минулого століття. І хоча з тих пір вийшла не одна версія цієї мови, а кожна така редакція вносила свої певні зміни, основні характеристики мови залишилися незмінними. Так даній мові програмування притаманна строга об'єктна орієнтованість, статична типізація та синтаксис, подібний до мови програмування C. Завдячуючи популярності самої мови та її широкій підтримці, Elasticsearch став дуже успішним проектом – базою даних, яку ще називають NoSQL через особливий спосіб зберігання даних. Справа в тому, що дана система управління базами даних зберігаю дані у неструктурованому вигляді, але при цьому можна використовувати звичайний синтаксис мови SQL для створення запитів до бази даних.

На відміну від інших схожих програм, Elasticsearch більше сфокусована на задачі пошуку даних, через це розробникам довелося створювати прикладний програмний інтерфейс, щоб підтримувати всі функції та можливості самої бази даних. Через те, що система розвивається не один рік, майже неможливо осягнути і знати в

деталях всі концепти та те, як працюють компоненти системи. Проте деякі базові концепти та терміни потрібно знати будь-якому користувачу для кращої взаємодії з даним продуктом.

Документи це об’єкти з даними, описаними з допомогою JSON. Використання JSON обумовлено тим, що така нотація дуже популярна в сучасних веб-сервісах, а для найпопулярнішої мови веб-програмування JavaScript така нотація ще й підтримується вбудовано. Тобто не потрібно перетворювати представлення з одного вигляду в інший – програма, отримавши відповідь від бази даних, автоматично перетворює нотацію та зберігає її в пам’яті.

Однією з найкращих рис пошукового рушія Elasticsearch [17] є його широкий програмний прикладний інтерфейс REST, який дозволяє швидку та просту інтеграцію з іншими веб сервісами, керування запитами, індексацію даних тощо. Взаємодія з таким інтерфейсом відбувається дуже просто, достатньо зробити HTTP запит та отримати відповідь у зручному для користувача форматі. Але оскільки Elasticsearch є складною системою з широким вибором функції та параметрів, важливо постійно слідкувати за оновленням документації, яка складається з окремих розділів, присвячених окремим темам:

- прикладний програмний інтерфейс документів Elasticsearch використовується для обробки документів. Наприклад, використовуючи цей інтерфейс можна створювати документи в індексі, оновлювати їх, переміщати до інших індексів або видаляти;
- прикладний програмний інтерфейс пошуку Elasticsearch використовується для запиту індексованих даних для пошуку та знаходження конкретної інформації. Інтерфейс можна застосовувати для всіх можливих індексів та типів, а в якості відповіді приходить документ з отриманими даними;
- прикладний програмний інтерфейс індексів Elasticsearch дозволяє адміністраторам та користувачам керувати індексами бази даних, відображеннями та шаблонами. Таким чином можна використовувати цей інтерфейс для створення або

видалення індексу, перевірки індексів на те, чи існує він в базі даних та визначення необхідного відображення для індексу;

- прикладний програмний інтерфейс кластерів Elasticsearch дозволяє створювати особливі кластерні запити для керування та контролю кластеру бази даних. В основному він використовується для того, щоб визначати, який вузол необхідно викликати, використовуючи ідентифікатор вузла або його ім’я та адресу.

Плагіни в середовищі Elasticsearch використовуються для розширення базової функціональності системи за допомогою створених користувачами підпрограм. Наприклад, користувачі можуть створити плагін, що збільшує безпеку програми або який додає механізми виявлення помилок та спрощення аналізу даних.

1.2.3.2 Огляд програмного забезпечення Logstash

Ефективний аналіз журналу логів та подій базується на добре структурованих логах, що не завжди легко досягнути. Структуровані дані спрощують процес пошуку інформації, дозволяють легше та швидше аналізувати та візуалізувати дані в будь-якому інструменті ведення логування, який використовує користувач. Структура це те, що дає контекст даних, який по можливості потрібно програмувати на рівні додатку, з якого надсилаються логи. Але в деяких випадках переведення в правильний формат надсилання даних на рівні програми неможливий. Саме тому існує інструмент Logstash, який в екосистемі стеку відіграє ключову роль в стандартизації та структуруванні даних.

Програмне забезпечення Logstash почало своє існування як інструмент з відкритим вихідним кодом, що розроблявся для обробки потокової передачі великої кількості логів з багатьох джерел одночасно. Після включення цієї програми як ключовий компонент екосистеми, він почав відігравати роль рушія, відповідального за обробку всіх вхідних повідомлень, їх структурування та приведення до потрібного для зберігання вигляду. Після виконання всіх операції ще одна ключова місія, за яку відповідальна дана програма – відправлення повідомлень до місця їх зберігання (зазвичай це Elasticsearch).

Завдяки широкому вибору плагінів, які встановлюються як підпрограми та розширюють функціонал самого програмного забезпечення Logstash, його можна також використовувати як інструмент для збору, перекодування та перетворення з одного вигляду даних на інший різних типів даних. Вже створено більше, ніж 200 різних плагінів для Logstash. Велика користувацька спільнота продовжує розширювати функціонал програми кожного дня.

Не зважаючи на конкуренцію з боку більш сильних гравців рину та певні недоліки в архітектурі розробленого додатку, Logstash все ж включили до складу екосистеми, а тепер він взагалі вважається одним з найважливіших компонентів стеку. Навіть з боку розробників були зроблені відповідальні кроки в питання вирішення цих проблем, вдосконаливши всю системи, створивши абсолютно новий механізм обробки запитів, який був продемонстрований у версії 7.0, що надало змогу вирватися вперед в боротьбі з конкурентами, зробило систему надійнішою та більш швидкою, ніж було раніше.

Одна з речей, що робить програму Logstash настільки популярною, успішною та потужною, – це її здатність агрегувати логи та події з різних джерел. Використовуючи більше 50 вхідних плагінів для різних платформ, кодувань та декодувань, баз даних та додатків, які можна встановити на сервер, щоб розширити функціонал та спростити процес збору та обробки даних з різних джерел та пересилання їх до інших систем для зберігання та аналізу.

Попри широку підтримку вхідних кодеків, дане програмне забезпечення підтримує ряд надзвичайно потужних плагінів фільтрів, які дозволяють збагачувати, змінвати вигляд та обробляти логи. Як і вхідні дані, Logstash підтримує ряд вихідних плагінів, які дозволяють надсилати свої дані в різні місця, служби та процеси. Можна зберігати результати обробки логів як файл, CSV та S3, перетворювати їх у повідомлення за допомогою RabbitMQ та SQS або відправляти їх у різні сервіси, такі як HipChat, PagerDuty або IRC. Кількість комбінацій входів і виходів у Logstash робить його дійсно універсальною програмою для обробки логів.

1.2.3.3 Огляд програмного забезпечення Kibana

Жодне рішення щодо централізованого ведення журналу логів не може бути прийнято без використання інструментів аналізу та візуалізації. Не маючи можливості створювати ефективні запити та відслідковувати дані, мало користі лише від агрегування та зберігання логів та подій. У екосистемі ELK Stack програмне забезпечення Kibana відіграє таку роль.

Kibana – це програма-користувальницький веб-інтерфейс на основі браузера, який може використовуватися для пошуку, аналізу та візуалізації даних, що зберігаються в індексах Elasticsearch. Це програмне забезпечення має відкритий вихідний код. А до недоліків можна віднести те, що Kibana не може бути використана разом з іншими базами даних. Особливо відомий і популярний цей додаток став завдяки багатим можливостям графіки та візуалізації, які дозволяють користувачам досліджувати великі обсяги даних візуально.

Пошук Elasticsearch конкретних повідомлень логів або рядків у цих повідомленнях – це спеціалізація цього програмного продукту. За замовчуванням користувачі тепер можуть використовувати нову мову для створення запитів в базу даних під назвою KQL (Kibana Querying Language) для пошуку даних. Серед найпоширеніших типів запитів виділяють такі:

- пошук вільного тексту – використовується для швидкого пошуку певного рядка з текстом;
- пошук на рівні поля – використовується для пошуку рядка в певному полі;
- логічні висловлювання – використовуються для об'єднання запитів в логічний вислів;
- неточний пошук – використовується для пошуку термінів в межах певної близькості висловлювань.

Через зручність у використанні, а також через розширені можливості графічного представлення даних для аналізу, даний веб-інтерфейс є ключовою частиною стеку для аналізу логів та метрик ELK Stack.

1.3 Аналіз існуючих алгоритмів та рішень управління якістю надання послуг критичними IT-інфраструктурами

Оскільки управління якістю надання послуг IT-інфраструктурами – це дуже важлива тема, то не дивно, що на ринку існує багато рішень, які справляються з таким завдання [13]. Деякі з рішень існують з моменту виникнення поняття «IT-інфраструктура», інші ж компанії, які займаються проблемою управління якістю набагато молодші, але надають можливість користуватися продуктами, які мають більш широкий функціонал, ніж свої конкуренти. Для того, щоб зрозуміти всі сильні та слабкі сторони таких продуктів, було проведено аналіз таких рішень, як SignalFx або DataDog.

1.3.1 Аналіз програмного забезпечення SignalFx

Програмне забезпечення SignalFx розширяє можливості Splunk. Саме по собі програмне забезпечення Splunk придатне в основному лише для збирання, зберігання та аналізу логів, але не має можливості управляти рівнем надання послуг IT-інфраструктурою. Разом з SignalFx поставляється сервіс для моніторингу інфраструктури, що має назву навігатор SignalFx для Kubernetes. Сервіс SignalFx працює як кінцева точка зі збору метрик та логів для їх подальшого аналізу.

Kubernetes Navigator являє собою ідеальну платформу для спостереження, моніторингу, аналізу та фільтрації середовищ, що містять контейнери, наприклад Kubernetes. SignalFx Kubernetes Navigator, схему якого зображено на рисунку 1.2, було розроблено у співпраці з провідними експертами з багаторічним досвідом роботи Kubernetes. Kubernetes Navigator пропонує рішення під ключ, що поєднує в собі потужність існуючою архітектури з цільовими візуалізаціями та аналітикою, які дозволяють оперативно вирішувати складні ситуації, не вимагаючи жодних налаштувань.

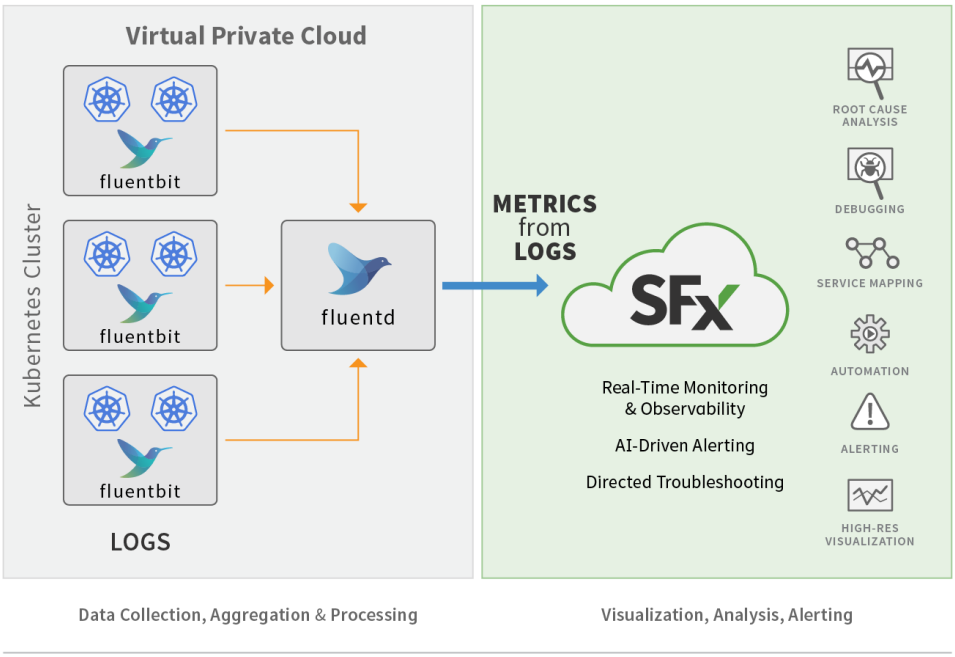

Рисунок 1.2 – Схематичне зображення компонентів системи SignalFx

Панелі навігації Kubernetes Navigator надають інформацію про об'єкти та характеристики Kubernetes, що дозволяє людині-оператору переходити між шарами абстракції та підтримувати ієрархічний контекст одразу після встановлення додатку. Це дозволяє високоефективно вивчати та досліджувати можливі проблеми. Щоб зробити тестування та вирішення проблем ще швидшими та активнішими, використовуються алгоритми штучного інтелекту, що постачаються з навігатором Kubernetes – вони автоматично аналізують проблему та надають рекомендації для подальшого вирішення проблеми.

Навігатор SignalFx Kubernetes надає операційним та інженерним командам розуміння, яке їм потрібно для моніторингу та управління здоров'ям контейнерних середовищ у режимі реального часу. Інженери зазвичай використовують SignalFx для нагляду за сервісами, які компанії переносять на Kubernetes, що надає операційній команді змогу швидко діагностувати будь-які проблеми з IT-інфраструктурою або самою платформою оркестрації.

Використовуючи заснований на відкритих стандартах розумний агент, SignalFx Kubernetes Navigator автоматично визначає повну ієрархію об'єктів та пов'язаних з

ними метаданих – кластерів, вузлів, под, контейнерів та просторів імен, а також робочих навантажень, що працюють на них. Оскільки ця інформація передається через платформу SignalFx, SignalFx Kubernetes Navigator динамічно створює інтерактивні карти кластерів, створює детальні списки вузлів та робочих навантажень та заповнює попередньо створені інформаційні панелі продуктивності.

SignalFx Kubernetes Navigator також аналізує дані в потоковому режимі, щоб миттєво дати користувачу знати про можливі помилки, згруповані за кластерними та іншими тегами метаданих, і збирає критично-необхідні дані, які можуть бути використані для співвідношення навантажень на сервіси, що працюють на об’єктах інфраструктури. Інтерфейс програми показано на рисунку 1.3

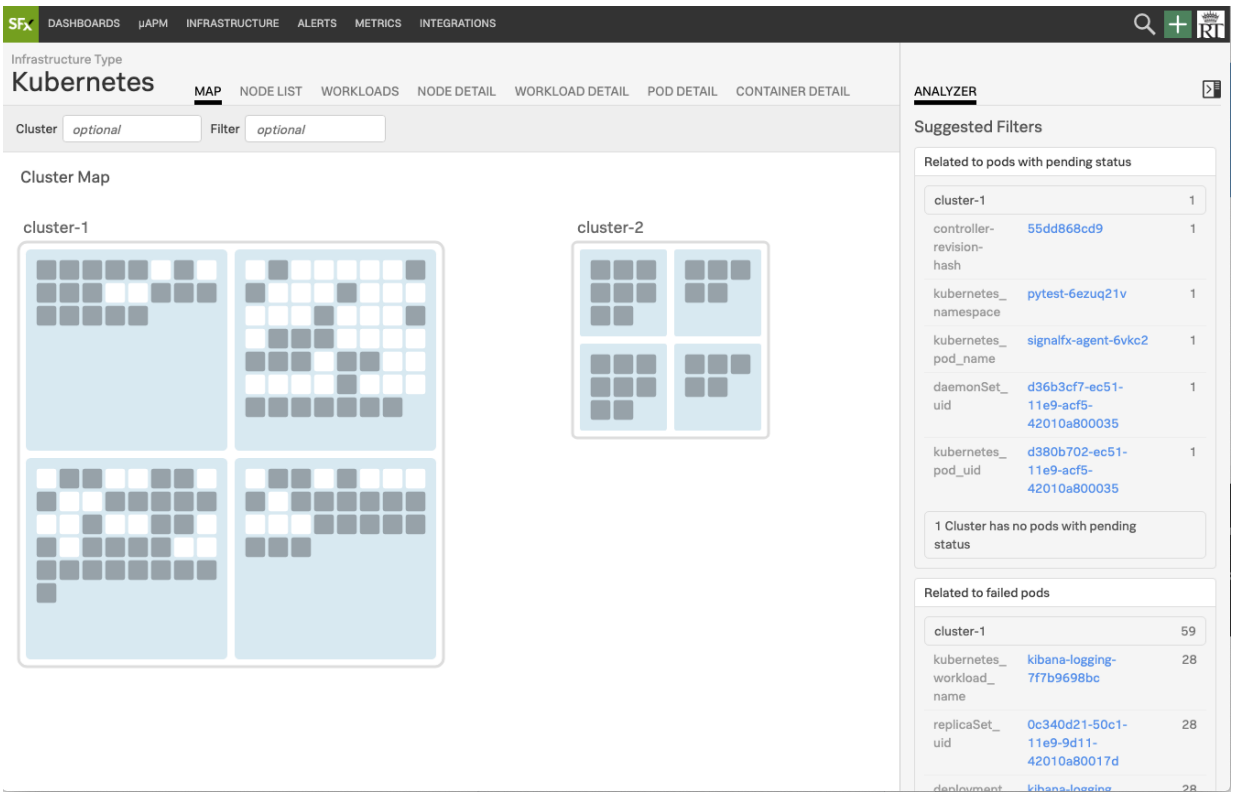

Рисунок 1.3 – Інтерактивна карта кластерів SignalFx Kubernetes Navigator

Користуючись перевагами аналізу в режимі реального часу, інтерактивним інтерфейсом користувачів та можливістю усунення несправностей з допомогою штучного інтелекту, користувачі можуть без зусиль збільшувати або зменшувати

масштаб серверів, фільтрувати та досліджувати найбільш навантажені середовища з контейнерами, щоб швидко помітити те, що до цього часу було важко знайти.

Крім того, маючи знання про навантаження на конкретних контейнерах, користувачі також можуть переходити прямо з навігатора SignalFx Kubernetes до SignalFx Microservices, щоб переглянути, зрозуміти та дослідити взаємозв'язок між різними об’єктами інфраструктури та службами, що працюють на них. Користувачі можуть використовувати різні метадані, такі як ідентифікатор контейнера, ідентифікатор робочого навантаження або ідентифікатор служби, щоб співвіднести те, як поведінка інфраструктури впливає на взаємодію сервісу та транзакції кінцевих користувачів або навпаки. Це особливо корисно під час усунення несправностей, коли командам DevOps потрібно швидко визначити, яка служба викликає раптові сплески затримки або частоти помилок і чому.

Навігатор SignalFx Kubernetes призначений для того, щоб надати відповіді на питання, які інакше було б складно і, в багатьох випадках, неможливо вирішити, використовуючи штучний інтелект для моніторингу. Відстеження ресурсів інфраструктури, потужностей та витрат, колись було простою задачею у хмарних середовищах. Оскільки інфраструктура організована на основі постійно мінливих потреб у ресурсах, операторам інфраструктури вкрай важко зрозуміти, які ресурси доступні в будь-який момент часу. Сильна сторона Kubernetes Navigator полягає в тому, що, орієнтуючи користувачів за допомогою динамічної та інтерактивної карти кластерів, дане програмне забезпечення надає доступ до аналітичної інформації. Карта допомагає візуально орієнтуватися користувачам в їх контейнерній інфраструктурі.

У динамічних та ефемерних середовищах Kubernetes власники сервісів часто стикаються з проблемами запуску або планування робочих навантажень. Вони також мають вирішувати проблеми, пов'язані з інфраструктурою, або проблеми з роботою під час звичайних операцій. У відповідь на ці питання власники послуг та оператори інфраструктури повинні зрозуміти, що змінилося. Наприклад, чи було розгорнуто новий контролер чи нову версію контролера. Чи було оновлено нову мережеву

підсистему. Зміни інфраструктури Kubernetes, навмисні чи ненавмисні, та додаткові навантаження від інших користувачів часто впливають один на одного. Kubernetes Navigator допомагає власникам сервісів та операторам інфраструктури проводити аналіз на рівні контейнерів, щоб миттєво виявити помилки оперативно.

Навігатор Kubernetes SignalFx забезпечує моніторинг у реальному часі та інтелектуальний аналіз середовища Kubernetes та робочих навантаженнь, надаючи власникам сервісів та операторам інфраструктури інформацію, яка їм потрібна для визначення проблеми та знаходження відповідного рішення заздалегідь, перш ніж ці проблеми вплинуть на клієнтів.

1.3.2 Аналіз програмного забезпечення Datadog

В основі програмного забезпечення для управління якістю надання послуг IT-інфраструктурами Datadog лежить програмне забезпечення Kubernetes, що породжує цікаві проблеми, оскільки Kubernetes забезпечує масштабування та збільшення ефективності системи. Щоб вирішити ці проблеми, розробник працювали над рішеннями, які допомогли б краще контролювати масштаб кластерів та полегшити розгортання та управління агентом Datadog. Дане програмне забезпечення надає можливість використовувати розширене трасування по кластерах Kubernetes.

Підтримка Datadog для розподіленого трасування за допомогою APM для Kubernetes надає користувачу глибоку прозорість запитів до кластерів, тож користувач зможе швидко визначити джерело будь-яких помилок та вузьких місць у навантажених контейнерами системах. Відстеження без обмежень дозволяє запитувати логи системи в онлайн режимі і зберігати їх для подальшого аналізу. І тепер кожен лог файл автоматично підписується метаданими контейнера – від окремих контейнерів аж до рівня розгортання та простору імен.

У вікні App Analytics користувач може відфільтрувати свої логи за будь-якою комбінацією фільтрів (наприклад, ім'ям контейнера, як показано нижче), і розглянути інформацію, щоб знайти запити, надіслані одним контейнером або процесом. Також можна натиснути на запит, щоб переглянути його графік в часі, який показує

тривалість життя запиту та час роботи кожного сервіса, який працював під час виконання запиту, як показано на рисунку 1.4.

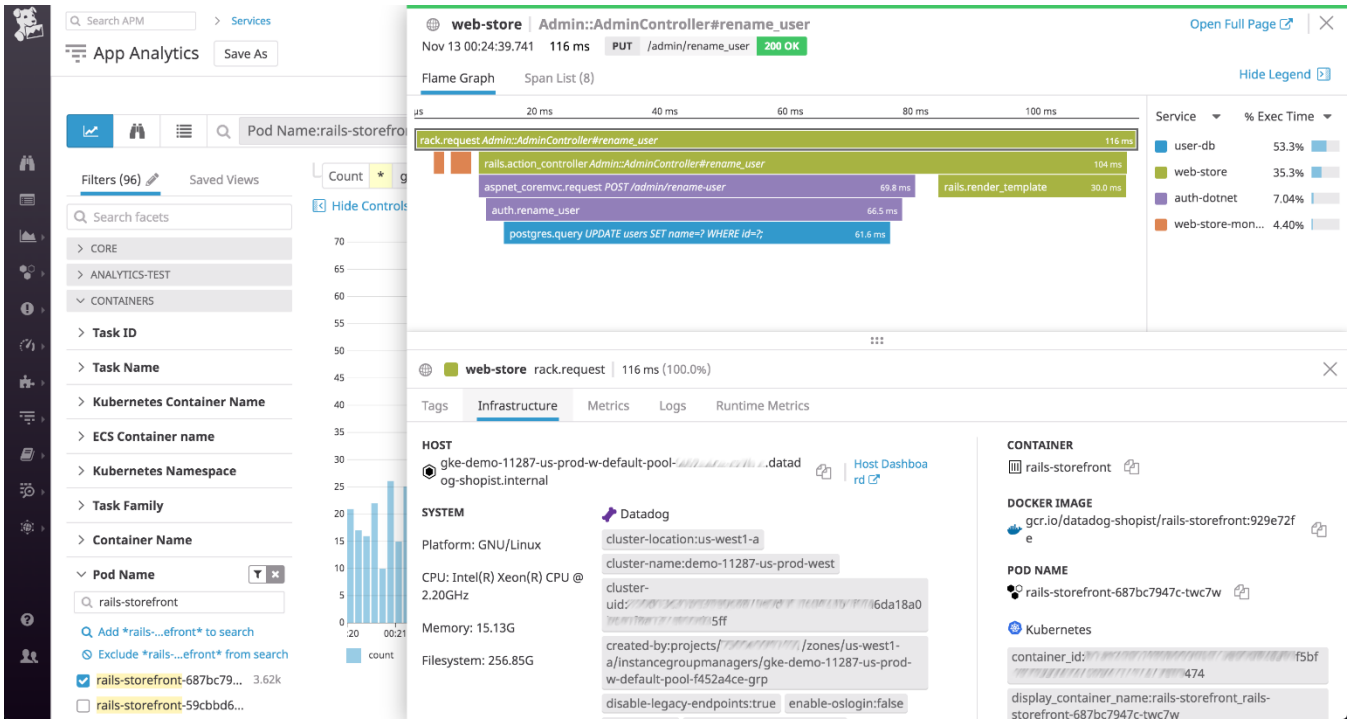

Рисунок 1.4 – Приклад фільтрації даних в Datadog

Часовий графік в Datadog показує часову шкалу запитів, які були виконані. Наведення курсору на проміжок часу, покаже теги хоста, який виконав цей запит. Адміністратор можете легко створювати сповіщення, наприклад, щоб повідомляти користувачів, якщо середня тривалість запитів, створених певним розгортанням, перевищує визначений поріг. Можна також налаштувати сповіщення для надсилання повідомлень на Slack, PagerDuty або будь-який інший інструмент співпраці, щоб команда змогла швидко розпочати розслідування причини затримки.

Також можна аналізувати логи безпосередньо в інтерфейсі Live Container. Достатньо натиснути на ім’я будь-якого контейнера, щоб переглянути його логи. У вікні «Контейнери» відображається список кластерів Kubernetes та відображаються показники процесора, пам’яті та мережі з кожного з них.

Показники розподілу агрегують дані моніторингу з кількох джерел (наприклад, контейнерів і подів) і обчислюють глобальні значення процента, що ілюструють

ефективність ваших послуг в цілому, що дозволяє глибше аналізувати досвід роботи користувачів. Можна візуалізувати та оповіщати про показники розповсюдження та використовувати їх для відстеження ефективності відповідно до угоди про рівень обслуговування, важливих для бізнесу.

Формат даних OpenMetrics використовується в основному багатьма хмарними технологіями, включаючи Kubernetes. Datadog включає також підтримку формату зберігання даних Prometheus та OpenMetrics. Також підтримується перетворення даних гістограми OpenMetrics в метрику розподілу, так що користувач може легко відслідковувати показники Kubernetes у вигляді відсотків у Datadog. КОрисутвач також може використовувати показники розповсюдження, щоб швидко зрозуміти ефективність служб у порівнянні зі службовими повідомленнями до команди.

Проте екосистема Datadog розвивається й за допомогою створення плагінів та додатків для інтеграції в основний програмний продукт. Розширення Watermark Pod Autoscaler (WPA), яке зараз знаходиться в бета-версії, має на меті розширити можливості горизонтального автоматичного масштабування под. Ще одне програмне забезпечення ExtendedDaemonSet (EDS) знаходиться в альфа-версії, воно розширює Kubernetes DaemonSet для вдосконалення процесу розгортання з функціями для випуску оновлень.

Розширення Watermark Pod Autoscaler – це проект з відкритим кодом, який розширює можливості основного продукту, для отримання більшого контролю над автоматичним масштабуванням кластерів для управління якістю надання послуг. HPA може масштабувати вашу інфраструктуру вертикально як в більшу сторону, так і в меншу та на основі встановленого метричного порогу, такого як середнє використання процесора для всіх под кластеру. Але коли HPA масштабується вгору або вниз, це може спричинити значну зміну значення показника якості системи та створити цикл проблем з масштабуванням. Наприклад, якщо визначено поріг масштабування як 40-відсоткове середнє використання процесора у всіх подах, а навантаження генерує 45-відсоткове використання на початкових двох подах, HPA додасть поди, внаслідок чого середнє значення використання процесору знизиться до

30 відсотків (або навіть нижче, якщо додано кілька под). Це нижче значення використання незабаром призведе до того, що HPA видалить поди, і в кластері знову буде недостатньо ресурсів.

Алгоритм WPA розширює алгоритм HPA, дозволяючи визначити діапазон прийнятних значень метрик (верхня і нижня межа), а не єдиний поріг, який HPA використовує для запуску масштабування. Наприклад, якщо встановити верхню межу в 40 відсотків середнього використання CPU у всіх подах, а нижню – у 20 відсотків, не відбудеться жодних масштабувань, поки середнє використання процесора залишається в цьому діапазоні. WPA додасть поди у набір реплік лише у тому випадку, коли середнє використання процесора перевищить 40 відсотків, і почне видаляти поди лише у випадку, якщо цей показник опуститься нижче 20 відсотків.

На графіку на рисунку 1.5 представлені верхня та нижня межі WPA у вигляді горизонтальних ліній. Якщо показник використання ресурсів знаходиться між цими значеннями, масштабування не відбудеться. Але якщо метрика рухається вище або нижче цього діапазону, кластер масштабується відповідно вгору або вниз і показник повертається в потрібний діапазон.

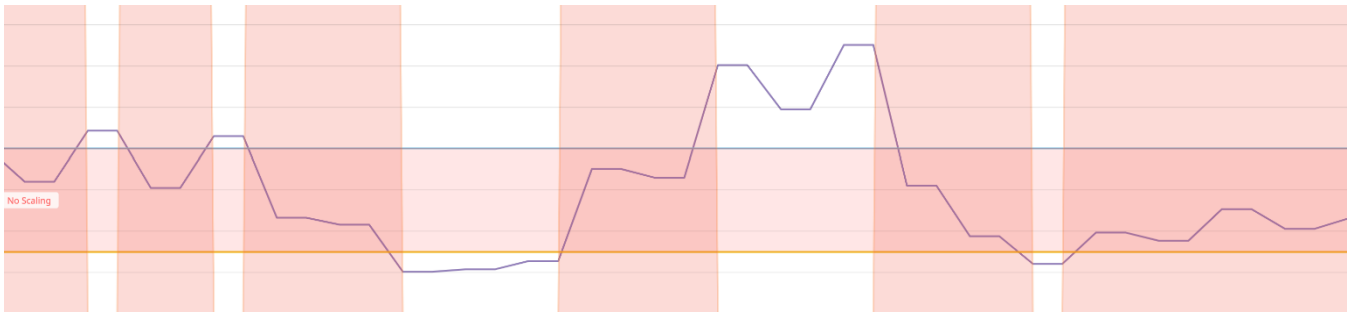

Рисунок 1.5 – Верхня та нижня межі використання ресурсів системою

Даний графік показує величину використання ресурсів в часі, починаючи з моменту, коли показник знаходиться в допустимих межах використання ресурсів. Потім показник переміщується назад у межі, потім нижче нижньої межі, а потім назад у межі. Крім того, WPA масштабує навантаження більш ефективно, дозволяючи

обмежити кількість реплік для додавання або видалення в рамках масштабування (у відсотках від поточного розміру кластера).

Це може захистити систему, наприклад, від занадто великого масштабування перед перерахунком потрібної кількості реплік. Можна визначити період виходу з ладу, щоб уникнути активації декількох процедур масштабувань підряд. За допомогою цих функцій WPA може керувати впливом будь-якого окремого процесу масштабування на розмір набору реплік, і таким чином можна уникнути раптових великих змін масштабів інфраструктури.

Щоб полегшити розгортання та налаштування агентів Datadog у інфраструктурі Kubernetes, був створений оператор Datadog. На основі шаблону Kubernetes Operator, Datadog Operator додає функціональні можливості, які допомагають керувати агентами Datadog. Налаштування конфігурації цих Агентів легко розробити в невеликому масштабі системи, але це стає більш трудомісткою задачею при великій та ефемерній інфраструктурі.

Наявність відкритого коду, підтримки з боку розробників та громади роблять такий продукт, як Datadog хорошим інструментом для управління якістю надання послуг критичною IT-інфраструктурою. А наявність плагінів, які надають можливість системі керувати ресурсами в режимі реального часу лише збільшує корисність такого додатку.

1.4 Висновки до розділу

В даному розділі на основі проведеного аналізу було визначено основні компоненти, з яких має складатися будь-яка системи управління якістю надання послуг критичними IT-інфраструктурами: інструменти для збору логів та метрик, сховище даних, та програмне забезпечення, що відповідає за візуалізацію даних та аналітику. Але окрім цих компонентів, важливу роль відіграє механізм управління рівнем надання послуг критичними IT-інфраструктурами.

Було визначено основні проблеми управління якістю надання послуг критичними IT-інфраструктурами – це недостатнє поширення підходу предиктивного моніторингу стану системи за допомогою механізмів збору логів та метрик. Такі дані можуть не лише допомагати швидко знаходити причину помилки, а з використанням правильних інструментів ще й попереджувати можливість виникнення критичних ситуацій у майбутньому.

Було визначено основні ресурси керування в критичній IT-інфраструктурі – це процесорний ресурс, кількість оперативної пам’яті та кількість пам’яті на диску. Було також визначено основні метрики, які впливають на якість надання послуг критичними IT-інфраструктурами – це в першу чергу швидкість відгуку системи, а також швидкість реагування відповідальної команди на критичні ситуації в системі.

В даному розділі було проведено порівняння, програмного забезпечення для збору та аналізу логів – ELK Stack та Splunk. Серед переваг ELK Stack зокрема це наявність відкритого програмного коду, широка підтримка сторонніх розширень та велика кількість форматів даних, які здатне оброблювати це програмне забезпечення. До недоліків можна віднести складність встановлення, адже система складається з трьох окремих компонентів, кожен з яких потрібно встановлювати окремо. Програмне забезпечення Splunk, має ряд переваг: простота у встановленні та використанні і зручний графічний інтерфейс. Проте суттєвими недоліками такої системи є її закритість та наявність платної підписки за використання. Тому було встановлено, що система ELK Stack набагато краще підходить для задачі моніторингу логів в критичних IT-інфраструктурах.

Під час аналізу рішень для управління якість надання послуг критичними інфраструктурами, було обрано два найбільш відомих на ринку програмних забезпечення – це Datadog та SignalFx. Було встановлено, що переваги кожного – це предиктивний моніторинг стану системи за допомогою евристик, проте ці системи розраховані на використання лише з системою Kubernetes, а по-друге, обидві системи мають закритий програмний код. Тому розробка системи управління якістю надання послуг критичними IT-інфраструктурами залишається відкритою задачею.

2 РОЗРОБКА ПІДСИСТЕМИ УПРАВЛІННЯ ЯКІСТЮ НАДАННЯ ПОСЛУГ СИСТЕМИ УПРАВЛІННЯ ЯКІСТЮ НАДАННЯ ПОСЛУГ КРИТИЧНИМИ ІТ-ІНФРАСТРУКТУРАМИ

2.1 Розробка структурної схеми підсистеми аналізу якості надання послуг критичними ІТ-інфраструктурами

Один з найважливіших етапів розробки системи – це розробка структурної схеми. Структурна схема – це візуальне зображення системи. Такий підхід використовує прості мічені блоки, які представляють собою окремі елементи або декілька елементів, сутностей або понять, з'єднаних лініями, щоб показати зв’язки між ними. Діаграма взаємозв'язків сутностей, один із прикладів структурної схеми, являє собою інформаційну систему, показуючи взаємозв'язки між людьми, об'єктами, місцями, поняттями або подіями всередині цієї системи.

Оскільки загалом структурні схеми широко використовуються при розробці та проектуванні в областях електроніки, проектування апаратних засобів, програмного забезпечення та процесів, то було вирішено використовувати саме такий підхід для розробки структури системи. Найчастіше такі схеми представляють поняття та системи на більш високому рівні та в менш детальному огляді. Діаграми корисні для вирішення технічних проблем. Структурні схеми є узагальненим поданням поняття і не призначені для відображення повної інформації щодо проекту чи виготовлення. На відміну від схем, креслення та схеми компонування, структурні схеми не відображають необхідних деталей для фізичної побудови. Такі схеми робляться зазвичай простими.

Спрощені структурні схеми також можуть бути корисними при демонстрації ідеї, але вони можуть приховувати внутрішню роботу потенційно секретної інтелектуальної власності. Після додавання достатньої кількості деталей за допомогою ітерацій, структурна схема стає схематичною. Використання структурної схеми призводить до модуляризації системи на високому рівні, яка розділяє загальну систему на максимально роз'єднані підсистеми. Структурні схеми системи

дозволяють візуалізувати систему як великі компоненти, що взаємодіють, які можна концептуалізувати та розробити самостійно. Цей тип архітектури також надає більшої гнучкості та розширюваності системи, що дозволяє їй рости і розвиватися легше підлаштовуватися під мінливі вимоги та вимоги. Тому розробка структурної схеми на початку процесу розробки є критично важливою для подальшої роботи над проектом.

Оскільки в даній роботі за об’єкт керування було обрано критичну ІТ-інфраструктуру, то очевидно, що один з компонентів структурної схеми має бути критичною ІТ-інфраструктурою. Також можна виокремити три інші компоненти системи. По-перше, це власне сервер системи управління якістю надання послуг. По-друге, це сервер візуалізації даних. І по-третє, це користувацький веб-інтерфейс. Розглянемо більш детально кожну з цих частин.

2.1.1 Розробка структурної схеми критичної ІТ-інфраструктури підприємства

Хоча в даній роботі було вирішено розглядати сервіси відкритих державних онлайн аукціонів як критичну ІТ-інфраструктуру, але схематичне зображення критичної ІТ-інфраструктури на структурній схемі не обмежується лише онлайн аукціонами. Справа в тому, що схематичне зображення критичної ІТ-інфраструктури, що складається з серверів та баз даних, які з’єднані в одну мережу – це досить загальне зображення критичної ІТ-інфраструктури [14].

Як показано на рисунку 2.1, критичні ІТ-інфраструктури можуть складатися з багатьох компонентів, при тому, що таким критичним системам зазвичай притаманна надлишковість. Як було зазначено вище, надлишковість може мати декілька проявів – починаючи від геонадлишковості, тобто рознесення даних та серверів на декілька географічно віддалених ЦОД, закінчуючи програмною надлишковістю. На прикладі критичної ІТ-інфраструктури підприємства, що займається онлайн аукціонами, можна продемонструвати прояв надлишковості системи.

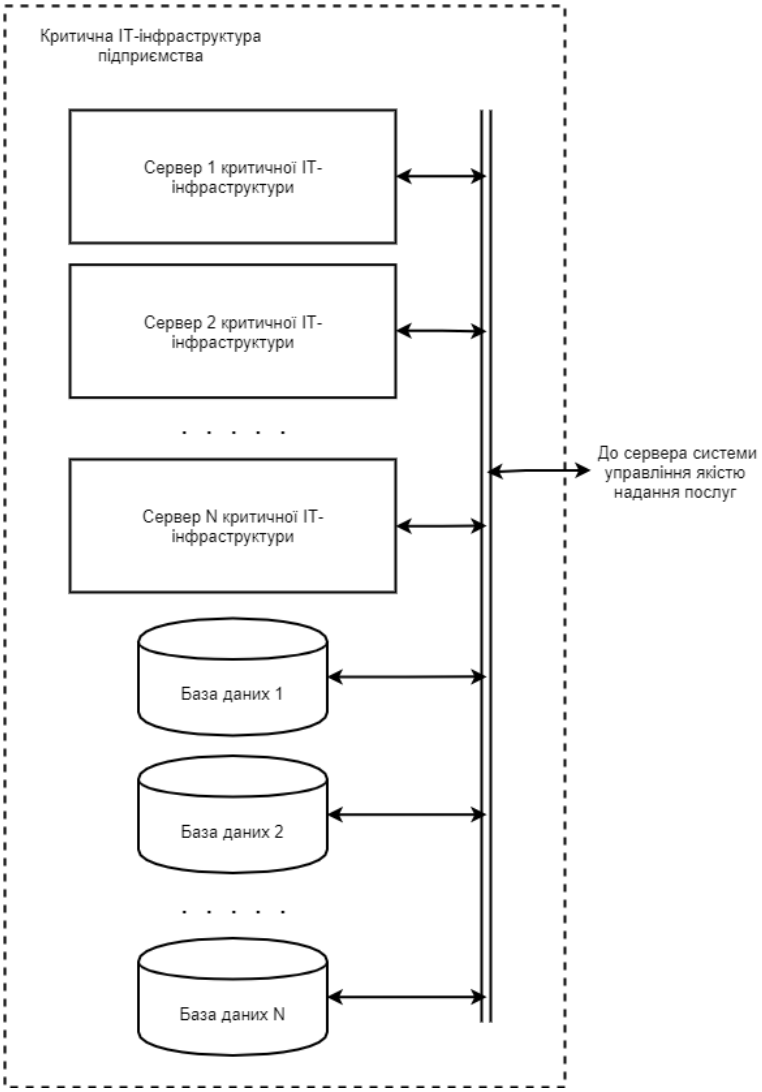

Рисунок 2.1 – Структурна схема критичної IT-інфраструктури

Задля забезпечення безперебійної роботи сервісів, кожен фізичний сервер зазвичай має ідентичний собі сервер для резервування. Техніка резервування дозволяє переносити дані та запускати сервіси на ідентичних серверах в разі, якщо один з сереверів-близнюків вимкнеться. В такому разі оскільки на сервері для резервування знаходяться найновіші копії даних та програмного забезпечення, час простою системи стає мінімальним.

Реплікація – одна з технік захисту масштабування баз даних, яка допомагає захищати систему від несподіваних зупинок серверів баз даних. Полягає ця техніка в тому, що дані з одного сервера бази даних постійно копіюються (реплікуються) на один або кілька інших. Для додатка з'являється можливість використовувати не один

сервер для обробки всіх запитів, а кілька. Таким чином з'являється можливість розподілити навантаження з одного сервера на кілька, а дані будуть зберігатися на декількох серверах одночасно.

Існує декілька підходів до реплікації баз даних. У першому з підходів виділяється один основний сервер бази даних, який називається головним. На ньому відбуваються всі зміни в даних (будь-які запити мовою SQL – INSERT / UPDATE / DELETE). Інший сервер, що називається підданим, постійно копіює всі зміни з головного. З додатків на підданий сервер відправляються запити читання даних. Таким чином головний сервер відповідає за зміни даних, а підданий сервер відповідає за читання даних. Перевага цього типу реплікації в тому, що з’являється можливість використовувати більше одного підданого сервера. Зазвичай слід використовувати не більше 20 таких серверів при роботі з одним головним сервером. В такому разі програмно вибирається випадковим чином один з підданих серверів для обробки запитів, тим самим розподіляючи навантаження на БД.

При виході з ладу одного з другорядних серверів, досить просто переналаштувати всі додатки на роботу з головним сервером. Єдиним шляхом до відновлення стану системи є реплікація на другорядний сервер і повторний його запуск. Якщо виходить з ладу головний сервер, тоді потрібно переключити всі операції (і читання і запису) на другорядні сервери. Таким чином один з другорядних серверів стане новим головним сервером. Після відновлення попереднього головного сервера, є можливість налаштувати на ньому реплікацію, і він тоді стане новим підданим сервером.

Деякі системи управління базами даних взагалі не мають вбудованої реплікації. У таких випадках, слід використовувати самостійну реалізацію реплікації. У найпростішому випадку, додаток буде дублювати всі запити відразу на кілька серверів бази даних. Реплікація використовується в більшій мірі для резервування баз даних і в меншій для масштабування. Такі реплікації зручні для розподілу запитів читання по декількох серверах. Підхід ручної реплікації дозволить використовувати переваги реплікації для технологій, які її не підтримують.

Слід зазначити, що реплікація сама по собі не дуже зручний механізм масштабування. Причиною тому – розсинхронізація даних і затримки в копіюванні з головного сервера на другорядний. Зате це відмінний засіб для забезпечення відмовостійкості. Завжди можна переключитися на використання другорядного серверу, якщо головний зупиняється і навпаки.

Резервування серверів з сервісами відбувається за схожим принципом. Існують сервери, на яких заздалегідь встановлені і налаштовані всі сервіси та додатки. В разі зупинки основного сервера, такий сервер може почати оброблювати запити майже одразу. Зазвичай маршрутизація запитів між такими серверами відбувається за допомогою проксі-сервера, який слугує абстракцією над прикладним програмним інтерфейсом обох серверів. Такий проксі-сервер також може виступати в ролі розподілювача навантажень, адже якщо декілька серверів з однаковим набором сервісів працюють паралельно, то кожен з них може обробляти запити користувачів паралельно.

Варто зазначити, що на схемі сервери критичної ІТ-інфраструктури пов’язані між собою певною централізованою шиною даних. До речі, такі мережеві з’єднання можна також резервувати. В основному це проявляється як прокладання паралельно декількох шин даних, хоча існує також багато інших підходів. Кожен сервер критичної ІТ-інфраструктури та кожна база даних під’єднані до локальної мережі, і всі вони можуть надсилати запити як один одному, так і запити в мережу інтернет.

Зовнішнє з’єднання дозволяє сервісам критичної ІТ-інфраструктури надсилати повідомлення на сервер системи управління якістю надання послуг критичними ІТ-інфраструктурами. Повідомлення надсилаються за допомогою протоколу HTTP прикладного рівня моделі OSI. В більшості випадків інформація про подій на серверах зберігається у вигляді JSON документів для зручності їх обробки в подальшому. Але є також можливість відправляти логи з використанням інших підходів форматування даних. Кожен з таких підходів налаштовється індивідуально на кожному з сервісів та підтримується агрегатором логів на сервері системи управління якістю надання послуг.

Програмний компонент, що використовується на сервіси критичної IT-інфраструктури задля збору та відправки логів також може бути встановлений користувачем довільно. Єдина вимога до такого компоненту це те, що він має підтримувати серіалізацію логів в такий вигляд, який є можливість оброблювати на сервері управління якістю надання послуг. Існує безліч програмних рішень, які мають широкий набір форматів, які підтримуються. Багато з таких продуктів мають відкритий вихідний код, що забезпечує простоту їх використання, інтеграції та налаштування.

2.1.2 Розробка структурної схеми сервера системи управління якістю надання послуг критичною IT-інфраструктурою

Сервер системи управління якістю надання послуг являє собою сервер, на якому встановлено систему з декількох модулів та бази даних, як показано на рисунку 2.2.

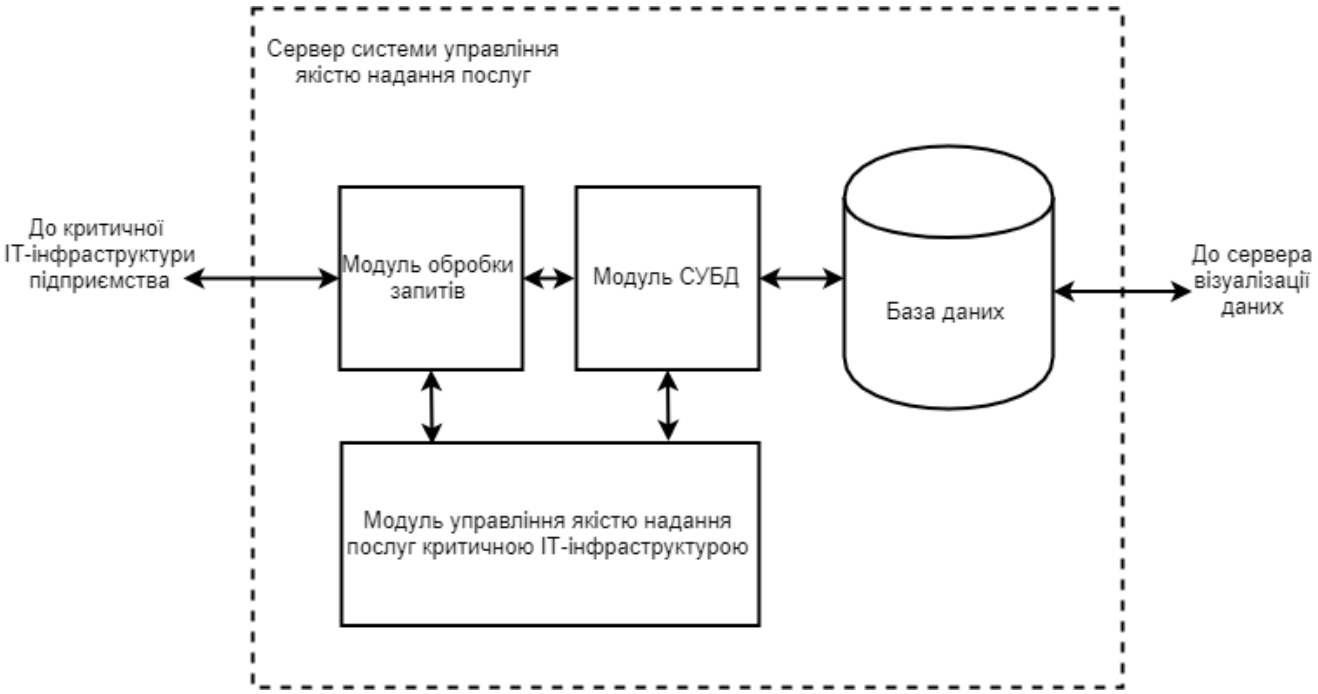

Рисунок 2.2 – Структурна схема сервера системи управління якістю надання послуг

Модуль обробки запитів являє собою програмний прикладний інтерфейс, який відповідає за отримання запитів від критичної IT-інфраструктури підприємства та надсилання відповідей на такі запити. В основному запити складаються з файлів логів та події, а основна задача модулю в такому випадку – це збереження даних до бази даних, надсилання запитів до сервісів критичної IT-інфраструктури від модулю управління якістю надання послуг, а також забезпечення безпеки системи.

Безпека є важливою частиною будь-якого процесу розробки програмного забезпечення. Навіть для публічного програмного прикладного інтерфейсу звичайна бізнес-вимога – це контроль над тим, хто може отримати доступ до послуг. Оскільки веб-API за правилами розробки інтерфейсів не зберігають стан системи, контекст безпеки не може залежати від сеансу роботи на сервері. Кожен запит, надісланий на прикладний програмний інтерфейс, повинен додавати певну форму облікових даних, яку необхідно перевіряти на сервері.

Аутентифікація HMAC – це механізм безпеки, який є загальним для загальнодоступних API та порівняно простий у застосуванні. Клієнт чи програма, яка хоче отримати доступ до послуги, потребуватиме секретний ключ від власника служби. Зазвичай ці ключі створюються випадковим чином, і заздалегідь надаються клієнту. Ключі API унікальні для кожного клієнта або програми. І клієнт, і сервер матимуть ключ API та секретний ключ.

Коли клієнт здійснює виклик в API, вміст повідомлення хешується за допомогою секретного ключа на клієнті для створення підпису HMAC. Це значення разом із оригінальним повідомленням та ключем інтерфейсу передається назад на сервер. На сервері той самий процес повторюється, але на цей раз за допомогою приватного ключа, що зберігається на сервері. Потім згенерований підпис HMAC порівнюється з тим, що був надісланий клієнтом. Запит вважається авторизованим, коли обидва підписи HMAC відповідають.

Цифровий підпис покладається на приватно-публічну пару ключів – зручний механізм для забезпечення безпеки між сервером та клієнтом. Цей підхід полягає в тому, що підписуючи вміст повідомлення приватним ключем, створюється захисний

підпис, який можна перевірити, використовуючи відповідний відкритий ключ-сертифікат. Пара ключів зазвичай надається відповідним органом сертифікації. Клієнт надає свій відкритий ключ серверу. Кожен запит подається на сервер, підписуючи вміст повідомлення за допомогою приватного ключа. Це дозволяє серверу перевірити справжність повідомлення, не знаючи приватного ключа клієнта. Відмінність цього методу від HMAC полягає в тому, що сервер і клієнт не поділяють один і той же секретний ключ.

Ще один з підходів до забезпечення безпеки передачі інформації в мережі це OAuth – популярний механізм безпеки, який широко використовується для аутентифікації користувачів. Подібно до того, як працює сеанс роботи на веб-сайті, OAuth вимагає від користувача клієнта "увійти" у веб-API, перш ніж надати доступ до решти сервісу. Це досягається шляхом викриття єдиної кінцевої точки для процесу входу. Клієнт передає свої облікові дані користувача в API, де відбувається автентифікація користувача на сервері. Після автентифікації генерується маркер безпеки, який зберігається на сервері та повертається клієнту. Кожного разу під час запиту клієнт передає цей маркер прикладного програмного інтерфейсу, щоб отримати доступ до кінцевих точок інтерфейсу. Маркер перевіряється на стороні сервера.

В процесі отримання логів з серверів критичної IT-інфраструктури безпека передачі даних відігає ключову роль. Отримавши дані від критичної IT-інфраструктури, модуль обробки запитів через модуль системи управління базами даних зберігає інформацію в базу даних у неструктурованому вигляді. Використання такого підходу обумовлено в першу чергу тим, що таким чином зменшується навантаження на сервер через відсутність додаткових обробок даних, що призводить до пришвидшення процесу збору та збереження даних.

Модуль управління якістю надання послуг критичною IT-інфраструктурою є одною з ключових складових компонентів системи. Даний модуль забезпечує управління якістю надання послуг через аналіз стану системи на поточний момент та використання моделей машинного навчання задля оцінки стану системи та

передбачення можливих подій в системі у майбутньому. Даний модуль має доступ до бази даних через модуль системи управління базами даних для отримання набору даних під час тренування моделі машинного навчання, а також під час роботи в режимі передбачення стану системи. Кожного разу, коли модуль отримує набір даних, який вказує на можливість виходу компоненту системи з ладу, відбувається запис інструкцій в базу даних через модуль СУБД, а також надсилається оповіщення до серверів критичної IT-інфраструктури.

База даних є центральним сховищем логів та подій, тому окрім доступу до даних в рамках сервера, також є можливість отримання доступу до бази даних з серверів, що розташовані в мережі інтернет. Одним з таких серверів є сервер візуалізації даних, що надсилає запити до центрального сховища даних задля створення візуалізацій та побудови аналітики.

2.1.3 Розробка структурної схеми сервера візуалізації даних

Сервер візуалізації даних, відіграє важливу роль в системі. Використання модулю системи управління базами даних надає можливість іншим модулям отримувати набори даних з бази даних сервера управління якістю надання послуг. Таким чином модуль агрегації даних може надсилати в базу даних запити задля отримання даних. Проблема полягає в тому, що в центральній базі логи та події зберігаються у вигляді окремих документів, деякі з яких ще й мають інакші протоколи серіалізації даних. Такий модуль слугує абстракцією над даними, дозволяючи приводити дані різного вигляду до даних такого вигляду, який було б зручно обробляти модулю візуалізації даних.

Модуль візуалізації даних використовує об’єднану попереднім модулем інформацію з бази даних для побудови різних типів графіків та візуалізацій. В найпростішому випадки це візуалізація накопичення даних, кількість отриманих системою логів за деякий проміжок часу, заданий користувачем. Для інших типів діаграм модуль має відповідні графічні інструменти для графічного відображення інформації.

Використання такого модулю дозволяє користувачу простіше сприймати інформацію, швидше аналізувати дані, виявляти помилки системи, а відповідно й швидше оброблювати такі події. Але самі по собі все ці модулю не працюватимуть без отримання запитів від модулю обробки запитів, який представляє собою програмний прикладний інтерфейс. Структурну схему сервера візуалізації даних зображено на рисунку 2.3.

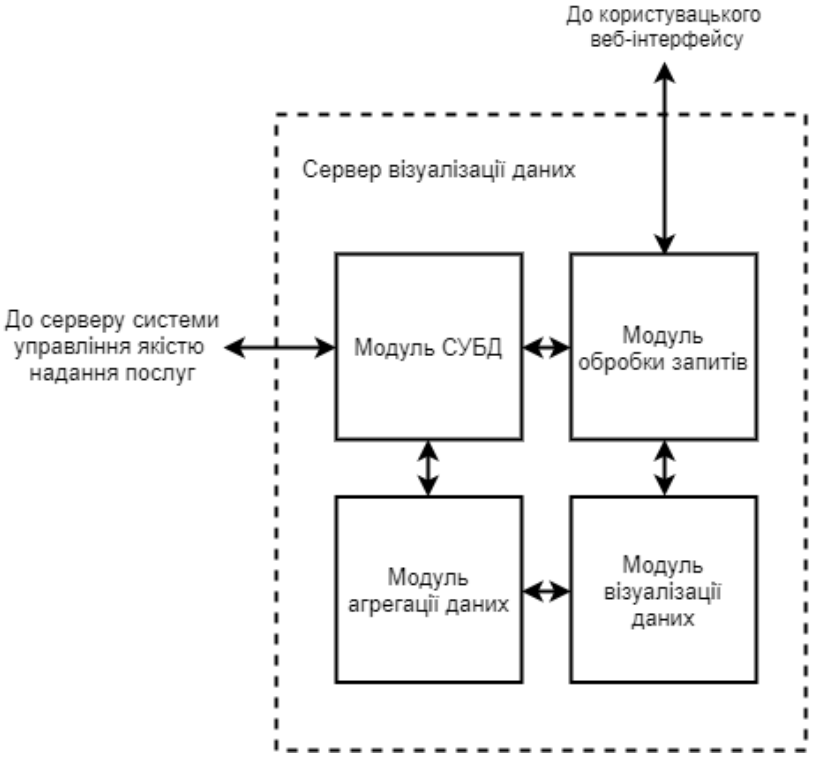

Рисунок 2.3 – Структурна схема сервера візуалізації даних

Даний інтерфейс є серверною частиною програмного забезпечення для візуалізації інформації, тому він має можливість отримувати запити від користувацького веб-інтерфейсу. Модуль обробки запитів може надсилати запити до бази даних через модуль СУБД, якщо потрібно надати користувачу витяг інформації з бази даних. Також існує можливістю надсилати користувачу у відповідь на запити візуалізацію даних, створену за допомогою модулю візуалізації.

2.1.4 Розробка структурної схеми користувацького веб-інтерфейсу

Користувацький веб-інтерфейс це ще одна частина системи, яка відповідає за візуалізацію інформації для користувача. Такий веб-інтерфейс може бути реалізований, як веб-додаток, що запускається через браузер користувача, як показано на рисунку 2.4. Такий веб-інтерфейс має зручний дизайн, що дозволяє користувачу економити час на взаємодію з ним, а також надає великий перелік інструментів для візуального аналізу даних: діаграми у вигляді стовпчиків, діаграми у вигляді кругів, так звані діаграми-водоспади, лінійні графіки, тривимірні фігури, графіки розподілу частот тощо. За візуалізацію відповідає модуль візуалізації даних, до якого користувацький веб-інтерфейс має доступ через надсилання HTTP-запитів до сервера візуалізації даних. Користувач також може використовувати даний візуальний веб-інтерфейс задля створення запитів в базу даних. В такому випадку дані будуть приходити в непідготовленому вигляді, але користувач все одно матиме можливість інтерактивно дослідити дані.

Рисунок 2.4 – Структурна схема користувацького веб-інтерфейсу

Для того, аби користувацький веб-інтерфейс дійсно відповідав вимогам, які було вказано вище, необхідно дотримуватися кращих практик для проектування інтерфейсів. Все випливає з того, що в першу чергу потрібно знати своїх користувачів, включаючи розуміння їх цілей, навичок, уподобань та тенденцій. Після

аналізу інформації про користувача, потрібно не забути врахувати наступні пункти при розробці інтерфейсу.

Такий веб-інтерфейс користувача потрібно зберігати простим. Кращі інтерфейси повинні бути майже непомітними для користувача. Потрібно уникати зайвих елементів і мати чіткі написи. Бажано створювати послідовність та використовувати загальні елементи інтерфейсу користувача. Використовуючи загальні елементи в інтерфейсі, користувачі відчувають себе комфортніше та зможуть швидше розібратися та почати користуватися інтерфейсом. Також важливо спочатку розробити дизайн на макеті.

Важливо також розглядати просторові зв’язки між елементами на сторінці та структурою сторінки важливості. Ретельне розміщення елементів може допомогти привернути увагу до найважливіших частин інтерфейсу та може допомогти користувачу швидше знаходити інформацію на сторінці та читати текст. Важливим є ще й використання кольору і текстури. Так можна спрямовувати увагу користувача на елементи, використовуючи колір, світло, контраст та текстуру.

Використання типографіки може піти на руку для створення ієрархії та чіткості. Слід бути уважним до того, як використовується шрифт. Різні розміри, шрифти та розташування тексту сприяють збільшенню масштабованості, розбірливості та читабельності. Користувачу також важливо, щоб система повідомляла про те, що відбувається. Особливо в задачі аналізу стану критичної ІТ-інфраструктури, завжди потрібно повідомляти користувачів про дії, зміни стану чи помилки на сервері. Різні елементи інтерфейсу використовуються для створення оповіщення про стан системи на даний момент та, якщо необхідно, надання інформації про наступні кроки. Ретельно продуманий дизайн надає змогу користувачу швидко орієнтуватися на сторінці та швидко реагувати на зміни в системі, що є дуже важливим в задачі управління якістю надання послуг критичними ІТ-інфраструктурами.

2.2 Розробка компонентів системи управління якістю надання послуг критичними IT-інфраструктурами

Розробка компонентів системи управління якістю надання послуг критичними IT-інфраструктурами це важливий процес створення системи, тому потрібно приділити особливу увагу цьому кроку та правильно налагодити процеси задля успішного виконання поставленої задачі. Для розробки компонентів системи було використано готові підходи до розробки програмного забезпечення, яких існує багато різновидів. Було вирішено використовувати підхід під назвою Agile.

2.2.1 Вибір гнучкої методології розробки програмного продукту Agile

Agile – це гнучкий підхід до розробки програмного забезпечення, що включає в себе різні методології (Scrum, Канбан, XP, Lean і інші). Agile – це ітеративний підхід до управління проектами та розробки програмного забезпечення, який допомагає командам швидше та з меншою кількістю головних болів приносити цінність для своїх клієнтів. Замість того, щоб робити все й одразу, Agile команда забезпечує роботу невеликими, але ітеративними кроками. Вимоги, плани та результати оцінюються постійно, щоб команди мали природний механізм швидкого реагування на зміни.

Зазвичай процеси розробки налаштовані в вигляді каскадної моделі (або waterfall model) – все відбувається поетапно і послідовно: задачі, виконані однією командою передаються на доопрацювання іншим командам. Команди ж гнучких підходів закликають до створення спільних міжфункціональних команд. Відкрите спілкування, співпраця, адаптація та довіра серед членів команди лежать в основі таких команд. Незважаючи на те, що керівник проекту або власник продукту, як правило, визначає пріоритетність роботи, яку потрібно виконати, команда бере на себе вирішення того, як буде виконана робота, самоорганізуючись навколо детальних завдань та завдань. Використання Agile-підходів до розробки допомогли за рахунок

своєї гнучкості мінімізувати всілякі ризики з допомогою набору принципів. Ці самі принципи і 4 основних ідеї зібрані в Agile-маніфесті, датованому 2001 роком:

- найголовніше люди, а не речі;
- документація не повинна нікому заважати працювати;
- співпраця, а не перечитування контракту.

Agile не визначається набором церемоній чи специфічних прийомів розробки. Швидше, agile – це набір методологій, який демонструє прихильність до жорстких циклів зворотного зв'язку та постійного вдосконалення продукту. Оригінальний маніфест Agile не передбачав двотижневих ітерацій або ідеального розміру команди. Він просто виклав набір основних цінностей, які ставлять людей на перше місце. Те, як команда працює сьогодні, повністю залежить від кожного члена команди.

На проектах гнучкий підхід до розробки зазвичай вибирають, щоб команди могли швидко реагувати на зміни на ринку або відгуки клієнтів, не збиваючи планів на рік. Досвід планування та доставки невеликими, частими кроками дозволяє команді збирати відгуки про кожну зміну та інтегрувати її у майбутні плани з мінімальними витратами. Насамперед, впровадження такої методології стосується самих людей. Як описано у маніфесті Agile, справжні людські взаємодії важливіші за жорсткі процеси. Співпраця з клієнтами та товаришами по команді важливіше за попередньо визначені домовленості. А надання робочого рішення проблеми клієнта важливіше, ніж написання дуже докладної документації.

Колектив об'єднується під спільним баченням, потім втілює його в життя так, як було заплановано ними. Кожна команда встановлює власні стандарти якості, зручності використання та повноти. Метрика готовності продукту потім інформує команду про те, наскільки швидко вони виконують роботу. Керівники компаній заявляють, що коли вони довіряють такій команді брати на себе відповідальність, така команда відчуває більше почуття відповідальності за продукт і старається задовольнити (або перевищити) очікування керівництва.

Публікація маніфесту Agile в 2001 році відзначає народження гнучкої методології. З тих пір з'явилося багато різновидів підходів до управління проектами,

таких як Scrum, Kanban, Lean та Extreme Programming (XP). Кожен втілює основні принципи: часті ітерації, постійне навчання команди та високу якість. Scrum та XP віддають перевагу командам з розробки програмного забезпечення, тоді як Kanban – поширений серед команд, орієнтованих на обслуговування, таких як сервіси підтримки IT або відділи управління людськими ресурсами.

Відкритість, довіра та автономія, яку команди отримують від використання таких методологій стають візитівкою для компаній, які хочуть залучити найкращих людей та отримати максимальну користь від них. Такі компанії вже доводять, що практика може відрізнятися в різних колективах, якщо вони керуються правильними принципами.

За Agile початок роботи над проектом починається з вибору власника продукту – людини, яка бачить, до якої мети йде проект і що хочеться отримати в результаті. По-друге, необхідно визначитися з командою – від 3 до 10 осіб, які володіють навичками, які дозволять отримати необхідний результат (тобто працездатний продукт).

Важливим є етап вибору скрам-майстра – людини, що стежить за ходом проекту і допомагає команді боротися з труднощами. Разом з такою людиною необхідно скласти беклог продукту – зібрати в одному місці (бажано на Agile-дошці) всі вимоги до продукту і розставити пріоритети. Власник продукту повинен продумати і зібрати всі побажання. Потім команда повинна оцінити беклог, щоб зрозуміти, чи можливо все це зробити і скільки часу буде потрібно.

Наступним етапом йде планування спринтів – відрізків часу (тиждень або два), за які команда виконує певний набір завдань. Спринти повинні бути регулярними, наприклад, 15 разів по два тижні, поки не вийде готовий продукт. Варто проводити щоденні зустрічі на 15 хвилин – на порядку три питання, на які коротко відповідає кожен: що було зроблено вчора, що буде зроблено сьогодні, і які перепони заважають іти далі. За підсумками спринту команда розповідає, що вдалося зробити, і демонструє працездатні частини продукту. На огляди може прийти хто завгодно: власник продукту, головний замовник або навіть потенційні клієнти. Після кожного

спринту Agile-команда обговорює проблеми і шукає рішення – це називається ретроспективою. Повинен вийти план змін, який команда відразу ж і запровадить – на наступному спринті.

Використання таких підходів дозволяє зменшити час на впровадження нових змін в продукти, а також допомагає зосередити увагу розробників на розробці конкретної частини продукту за раз, без потреби одночасно вирішувати багато задач. Це забезпечує гнучкість у впровадженні нових змін та водночас забезпечує постійний темп розробки.

2.2.2 Розробка структури угоди про рівень надання послуг

Окремо потрібно оглянути процес розробки структури SLA – це один з важливих етапів розробки IT-інфраструктури. Продуктивність постачальника послуг оцінюється відповідно до набору показників. Час відповіді та час вирішення проблеми є одними з ключових показників, що містяться у домовленості про рівень надання послуг, оскільки вони стосуються того, як постачальник послуг вирішує ситуації з недоступністю сервісів. Угоди про рівень надання послуг визначають очікування клієнтів щодо ефективності та якості постачальника послуг різними способами. Деякі показники, які SLA можуть включати в себе:

- відсоток доступності та часу безперебійного користування – кількість часу, протягом якого сервіси працюють та доступні для замовника. Час постійної роботи сервера, як правило, відстежується та повідомляється в кінці кожного календарного місяця;
- конкретні показники ефективності, з якими періодично буде порівнюватися фактична ефективність;
- час відповіді постачальника послуг, тобто час, який потрібен постачальнику послуг, щоб відповісти на питання або запит клієнта. Якщо постачальник послуг має достатньо ресурсів, він може керувати власною службою обслуговування, щоб відповідати на запити клієнтів;

- час вирішення проблеми, тобто час, який потрібен для вирішення проблеми після її реєстрації постачальником послуг;
- графік повідомлень про зміни в роботі мережі, які пов’язані з оновленнями та сервісними роботами і можуть вплинути на користувачів.

Загалом SLA може визначати доступність, продуктивність та інші параметри для різних типів інфраструктури клієнта – внутрішніх мереж, серверів та компонентів інфраструктури, таких як джерела безперебійного живлення.

Окрім встановлення показників ефективності, угода про рівень надання послуг може містити план вирішення проблем простою та документацію щодо того, як постачальник послуг буде компенсувати клієнтам збитки у разі порушення договору. Кредити на обслуговування – типовий засіб захисту. Тут постачальник послуг видає клієнту кредити на основі розрахунку, визначеного у додатковій угоді. Наприклад, постачальники послуг можуть надавати кредити, що відповідають кількості часу, протягом якого було неможливо отримати доступ до серверів. Постачальник послуг може обмежити штрафні санкції, вказавши, наприклад, максимальну суму у валюті, щоб обмежити санкції.

Існує кілька способів управління рівнем обслуговування, за допомогою яких можна структурувати свої угоди про обслуговування. ITIL зосереджується на трьох типах варіантів структуризації угод про обслуговування (SLA): сервісному, споживчому та багаторівневому або ієрархічному угод про надання послуг. При вирішенні питання про те, яка структура SLA є найбільш підходящою для організації, потрібно враховувати багато різних факторів. Наприклад, поділяють керування багаторівневої структури SLA на компоненти.

На корпоративному рівні висвітлюються усі загальні питання, що стосуються організації, і вони однакові у всій організації. Прикладом узгодження рівня надання послуг на рівні організації може слугувати правило, за яким кожен співробітник повинен створити паролі з 8 символів і повинен змінювати їх кожні тридцять днів – або кожен працівник повинен мати картку доступу з біометричною інформацією.

На рівні клієнта розглядаються ті питання, які специфічні для кожного замовника. Наприклад, фінансовому департаменту потрібні більш високі заходи безпеки в силу його вирішальної ролі та управління фінансовими ресурсами. Вимоги до безпеки одного або декількох відділів всередині організації цілком можливо регулювати за допомогою вимог на рівні клієнта.

Останній рівень – це рівень послуг. Усі питання, що стосуються конкретної послуги (стосовно клієнта), можуть бути охоплені цим рівнем. Застосовується він до всіх клієнтів, які використовують одну і ту ж послугу – наприклад, укладання договору про послуги підтримки IT-відділу для всіх клієнтів, які використовують певного постачальника IP-телефонії.

Використання багаторівневої структури для великої організації запобігає дублюванню тих самих правил, одночасно забезпечуючи підлаштування правил під окремих клієнтів та послуг. Таким чином, угоди про рівень надання послуг корпоративними IT-інфраструктурами застосовуються до всіх і кожного відділу цієї організації, клієнтські угоди на рівні клієнтів застосовуються до відділу тощо.

2.3 Розгортання компонентів системи управління якістю надання послуг критичними IT-інфраструктурами

Через зростаючу потребу відслідковувати стан програмного забезпечення та IT-інфраструктури, на якій воно встановлене, ELK виявився саме тим рішенням, що надало можливість збирати та обробляти дані з багатьох джерел даних одночасно, зберігати ці дані в централізованому сховищі з підтримкою масштабування та аналізувати їх за допомогою вбудованих інструментів.

2.3.1 Розгортання компонентів ELK Stack на серверах IT-інфраструктури

Переваги в розгортанні ELK Stack – це в першу чергу відкритий вихідний код програми, що дозволяє швидко вирішувати будь-які проблеми при встановленні, адже громада, яка підтримує такий продукт з відкритим кодом, зазвичай приймає активну

участь у вирішенні проблем та відповіді на питання стосовно продукту. Екосистема складається з чотирьох основних компонентів, кожен з яких може бути доволі просто встановлений на сервери IT-інфраструктури підприємства.

Найпростіший спосіб встановити дане програмне забезпечення – це скачати відповідні програмні пакети для встановлення. Зазвичай для розгортання цієї системи використовуються сервери, що працюють на операційній системі Linux, але процес не обмежується лише нею. Проте варто зазначити, що хоча два основних продукти екосистеми і можуть бути встановлені через консоль, компонент Logstash зазвичай інтегрується в програмне забезпечення користувача та встановлюється як окрема бібліотека.

Одним з ключових понять системи управління базами даних є індекси. Індекси це логічні конструкції для розбиття документів на частини для швидшого пошуку інформації. Після встановлення Elasticsearch можна створити стільки індексів, скільки потрібно користувачу або адміністратору баз даних, але надто велика їх кількість призведе до того, що буде поступово втрачатися швидкодія. Індекси створюються за допомогою внутрішніх команд та ідентифікуються за допомогою унікальних імен, щоб було простіше їх використовувати в додатках та виконувати певні дії над ними, такі як видалення, пошук тощо.

Розміри індексів в Elasticsearch зазвичай можуть сягати десятків гігабайт на проектах з великою кількістю даних та оборотом даних, що може в свою чергу призводити до поломки та вимкнення додатку. Оскільки не існує обмеження на те, скільки документів можна зберігати в рамках одного індексу, то індекси можуть рости в розмірах неконтрольовано та врешті-решт відбирати місце на диску у бази даних. Коли на диску закінчується місце, це може призводити до відмови бази даних індексувати дані.

Один з підходів до боротьби з такою проблемою це розділення індексів горизонтально на окремі частини, що називаються осколками. Це дозволяє розподілити операції пошуку серед таких осколків на різних вузлах пошукової

системи та пришвидшить операцію пошуку. Можна налаштовувати кількість таких осколків на один індекс та зберігати їх на окремих вузлах системи.

За замовчуванням після встановлення база даних використовує концепт документів для зберігання даних. Якщо порівнювати реляційні та нереляційні бази даних, то документи можна порівняти зі строками в звичайних базах даних. Стандартний вигляд документа описується вже згаданим вище JSON. Дані зберігаються у вигляді пар ключ – значення, де ключем зазвичай буває поле типу строка, а даними можуть бути такі типи, як цілі числа, числа з плаваючою комою, строки, булеві змінні і навіть інші складні типи.

Також бажано налаштувати реплікацію даних одразу після встановлення продукту. Реплікація даних дозволяє адміністратору просто та зручно відновлювати стан системи після таких поломок, як неочікуване падіння системи або проблеми з мережею. Оскільки підхід реплікації був розроблений для того, щоб переконатися, що система є високо доступною, вони розміщуються на різних вузлах системи, тобто фізичних серверах, з яких вони потім копіюються на інші сервери. Схоже на те, як осколки, кількість реплік можна налаштовувати і змінювати потім.

Для запису даних в базу використовується рушій для збирання та обробки логів, що називається Logstash. Налаштування Logstash це один з найважливіших етапів встановлення стеку. Події, які агрегуються та обробляються за допомогою програмного забезпечення, проходять три основні етапи: збір, обробка та пересилання. Інформація щодо того, які дані збираються, як вони обробляються та куди надсилаються, визначено у файлі конфігурації, який знаходиться на диску. Кожен з цих етапів описаний та налаштований у файлі конфігурації в секції «Плагін». Так секція плагіну «Збір» визначає налаштування для етапу збору даних, секція плагіну «Фільтр» – для етапу обробки інформації та плагіну «Пересилання» для етапу відправки агрегованих та оброблених даних. І вхідні, і вихідні плагіни, і ті що розроблені та підтримуються громадою підтримують кодеки – особливе програмне забезпечення, що дозволяє кодувати або декодувати дані у певному вигляді.

Налаштування кодеків вхідних та вихідних даних є обов'язковою частиною процесу встановлення системи. Кодеки можна використовувати як на входах, так і на виходах. Вхідні кодеки забезпечують зручний спосіб декодування даних ще до їх збереження. Вихідні кодеки забезпечують зручний спосіб кодування даних, перш ніж вони пересилаються далі. Існує декілька найпоширеніших кодеків, такі як: звичайний текст без розмежування між подіями, кодек "json" призначений для кодування подій JSON на входах та декодування повідомлень JSON у вихідних даних, а режим "Rubydebug", дуже корисний при налагодженні, адже дозволяє виводити події Logstash як внутрішні об'єкти мови програмування Ruby. Налаштування необхідних кодеків відбувається через конфігураційні файли системи.

2.3.2 Розгортання серверу системи управління якістю надання послуг критичними IT-інфраструктурами

Відповідно до схеми, приведеної в додатку Д, розгортання бази даних Elasticsearch відбувається на окремому фізичному сервері системи управління базами даних, як показано на рисунку 2.5.

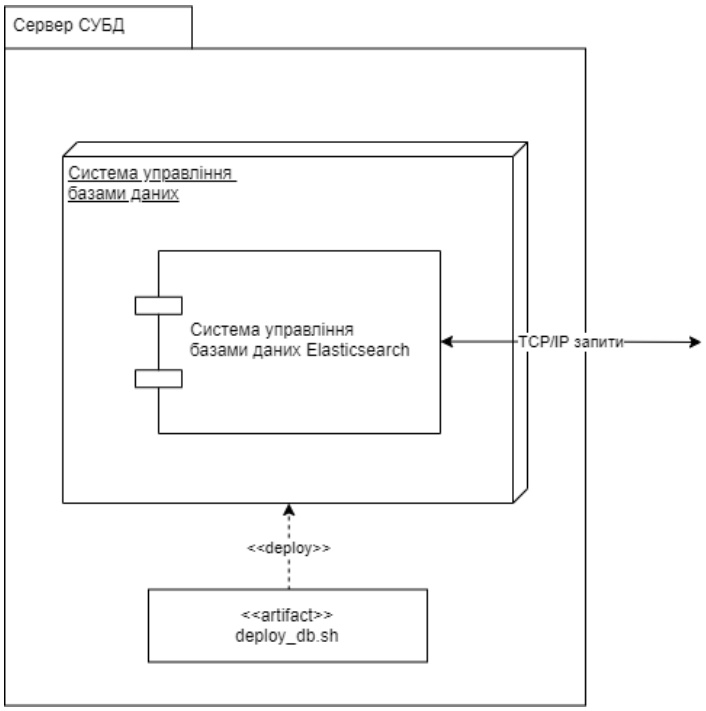

Рисунок 2.5 – Діаграма розгортання серверу СУБД

Для розгортання використовується автоматичний скрипт deploy_db.sh, який налаштовує середовище для СУБД – встановлює потрібні інструменти та бібліотеки, налаштовує правила системи безпеки та доступи, створює нового користувача для СУБД, встановлює програмне забезпечення, створює та запускає сервіс Elasticsearch.

Даний сервер може обмінюватися даними з сервером системи управління якістю надання послуг критичними IT-інфраструктурами за допомогою протоколів TCP/IP. В свою чергу, на фізичному сервері системи управління якістю надання послуг критичними IT-інфраструктурами, необхідно розгорнути три основні компоненти. В першу чергу встановлюється модуль, що містить логіку обробки запитів. Даний модуль являє собою веб програмний прикладний інтерфейс, який встановлюється за допомогою скрипту deploy_quality_management.sh на сервері, як показано на рисунку 2.6.

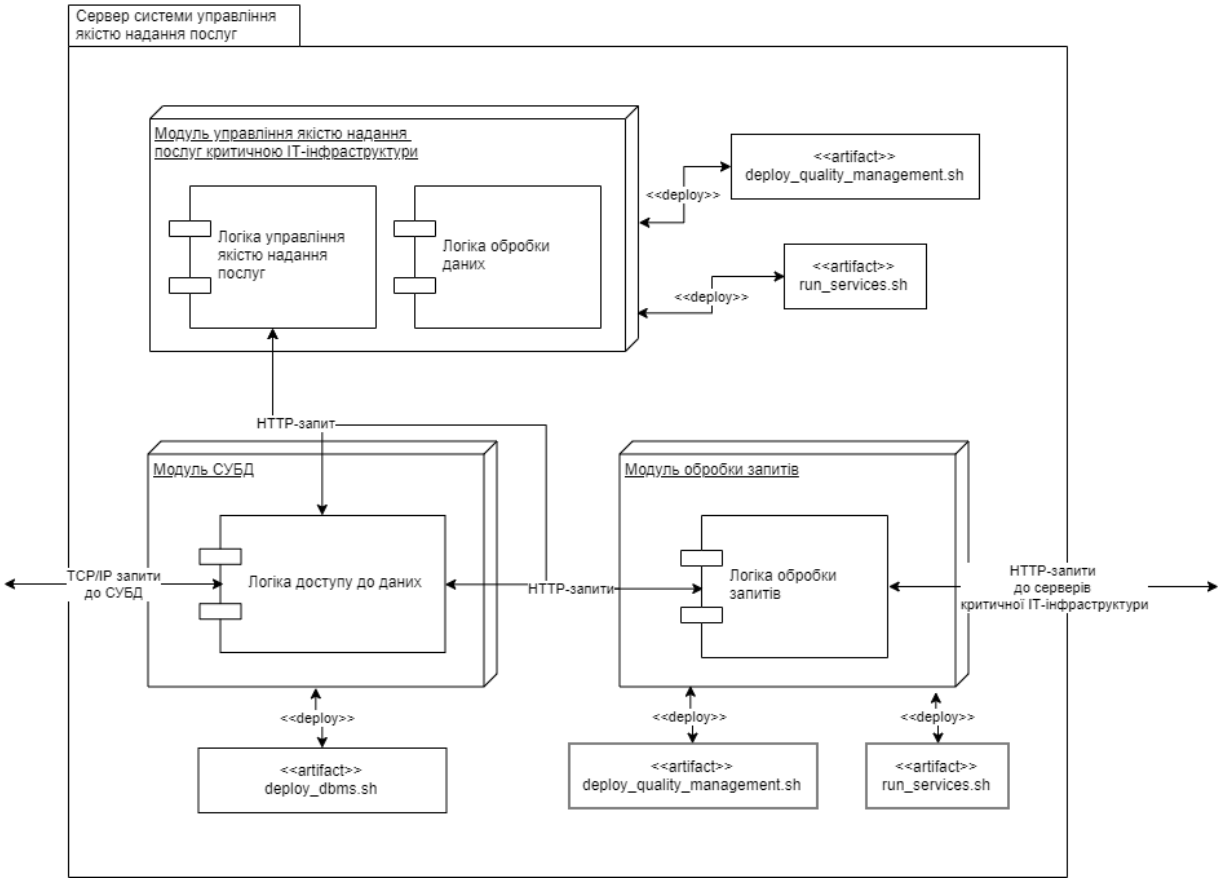

Рисунок 2.6 – Схема розгортання компонентів серверу системи управління якістю надання послуг критичними IT-інфраструктурами

Даний скрипт налаштовує середовище, аналогічно до скрипту розгортання СУБД. Після закінчення операції встановлення запускається скрипт run_services.sh, який запускає веб сервіси. Аналогічним образом встановлюються модулі управління якістю надання послуг критичною IT-інфраструктурою та модуль СУБД. На сервері також зберігаються відповідні скрипти для встановлення та запуску служб. Додатково модуль обробки запитів має можливість надсилати запити та отримувати запити HTTP від серверів критичної IT-інфраструктури. Наступним кроком відбувається налаштування графічного інтерфейсу.

2.3.3 Налаштування веб-інтерфейсу Kibana

Після встановлення програмного забезпечення Kibana, користувач має можливість відкрити веб-інтерфейс програми через браузер. Щоб покращити пошук у Kibana, при введенні пошукового запиту в строчці пошуку автоматично спрацьовує функція автодоповнення, яке надає можливість користувачу переглянути синтаксис пошуку під час введення запиту, як показано на рисунку 2.7.

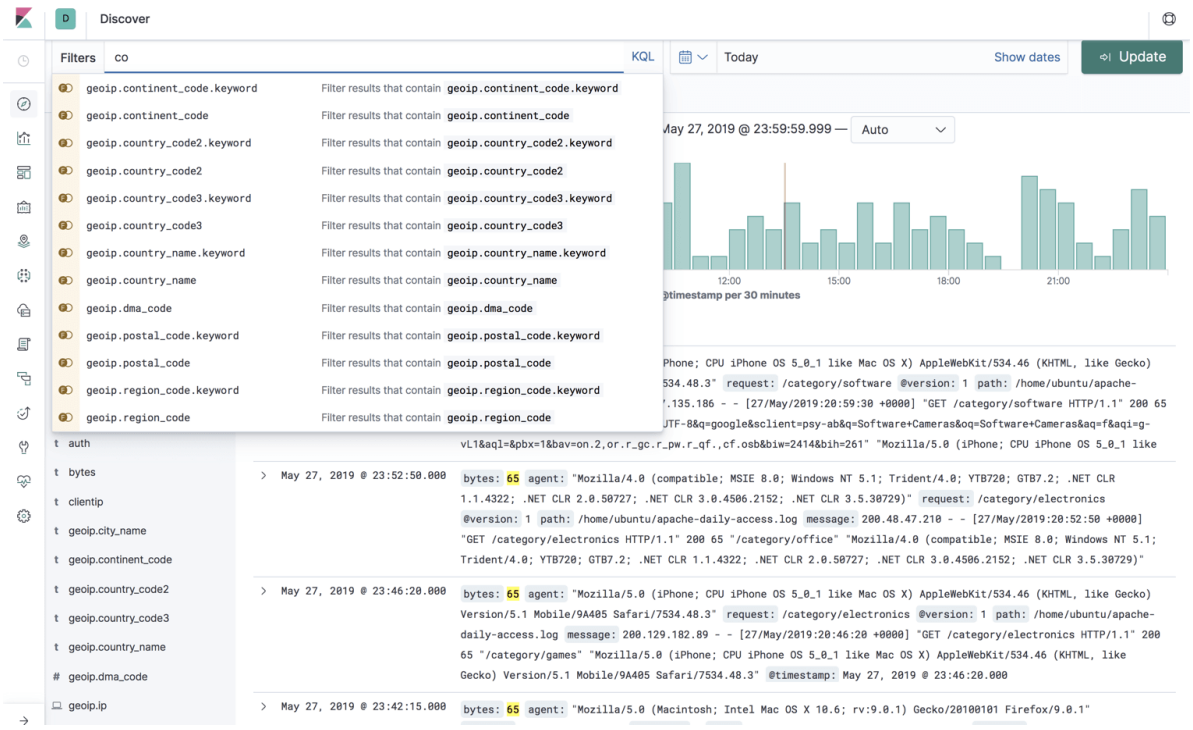

Рисунок 2.7 – Приклад автодоповнення вводу в Kibana

Щоб допомогти користувачам з пошуком, Kibana включає діалогове вікно фільтрації, яке дозволяє простіше фільтрувати дані, що відображаються в головному вікні, як показано на рисунку 2.8.

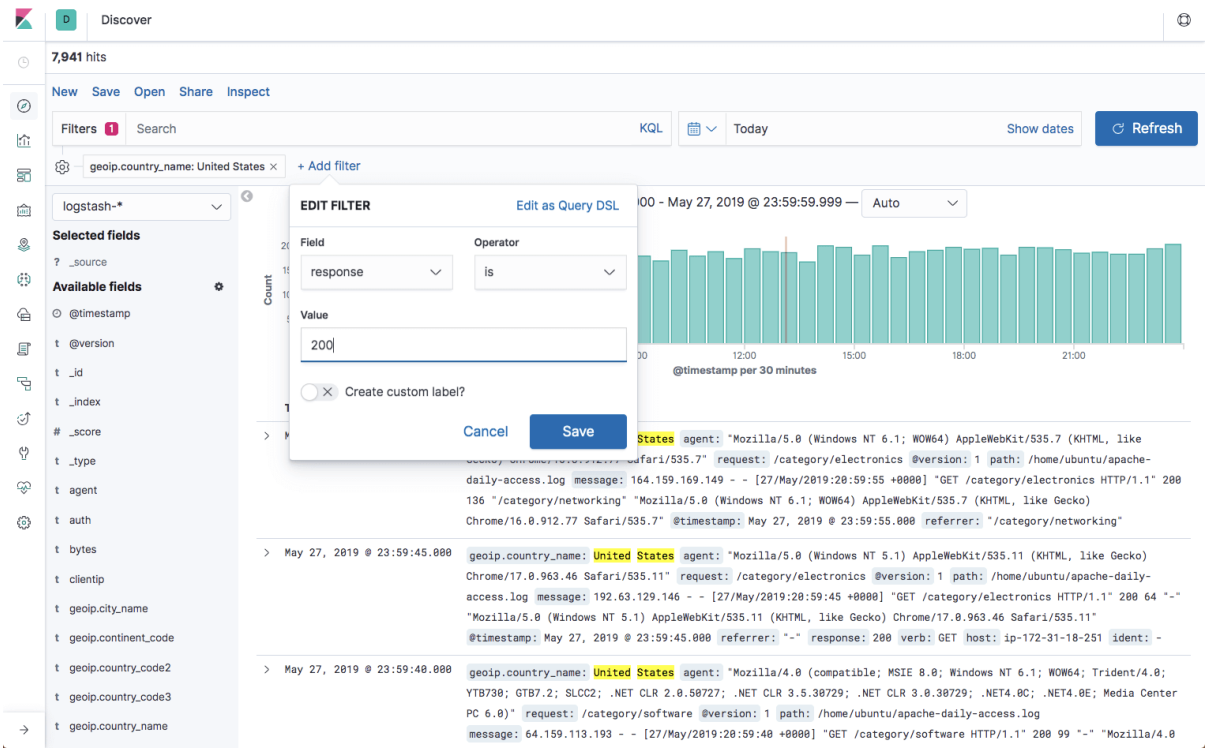

Рисунок 2.8 – Приклад фільтрації часових проміжків в Kibana

Як було зазначено вище, програмне забезпечення Kibana відоме своїми можливостями візуалізації. Використовуючи широкий вибір різних діаграм і графіків, користувач може візуально відтворити та проаналізувати свої дані будь-яким способом. Також надається можливість створювати власні спеціалізовані візуалізації за допомогою сторонніх інструментів.

Однак створення правильних та інтуїтивних візуалізацій це не завжди просто і може зайняти деякий час. Головне в цьому процесі – це розібратися в даних. Чим більше користувач ознайомлений з різними аспектами своїх даних, тим легше буде їх аналізувати. Для цього існують спеціальні візуальні об’єкти, які можна створити в Kibana, як показано на рисунку 2.9.

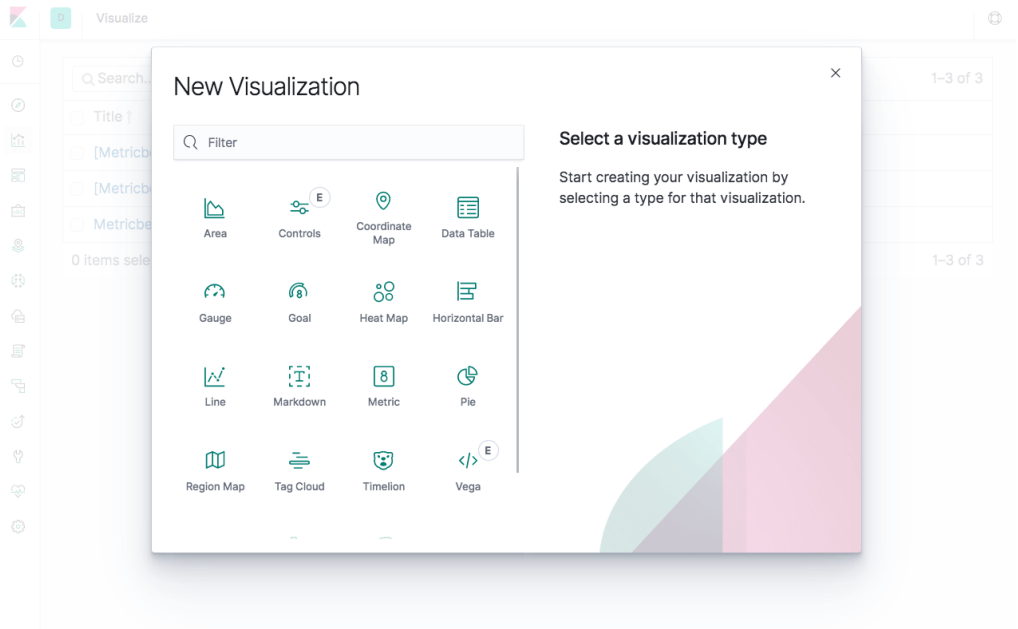

Рисунок 2.9 – Створення нової візуалізації в Kibana

Візуалізації Kibana побудовані на основі запитів до бази даних Elasticsearch. Використовуючи команди агрегації та синтаксис запитів (наприклад, сума, середнє, мінімальне, максимальне тощо), користувач може по-різному обробляти дані, щоб візуалізації відображали тенденції в даних. Але такі програмні продукти, як ELK Stack або Splunk мають один недолік з точки зору систем для управління якістю надання послуг критичними IT-інфраструктурами. Такі продукти самі по собі використовуються для візуального аналізу даних, але їх можливості щодо передбачення стану системи дуже обмежені.

Загалом робота адміністратора систем з підготовки системи до користування полягає в попередньому налаштуванні користувацького інтерфейсу та створення відповідних візуалізацій для спрощення процесу використання системи в подальшому.

2.4 Висновки до розділу

Отже, в даному розділі було розроблено структурну схему системи управління якістю надання послуг критичними IT-інфраструктурами. В структурній схемі було виокремлено чотири основні компоненти системи – це власне критична IT-

інфраструктура підприємства, яка складається з серверів додатків та баз даний, які можуть дублюватися для забезпечення стабільної роботи системи.

Було розроблено структурну схему для серверу системи управління якістю надання послуг критичними IT-інфраструктурами – сервер, на якому встановлено систему управління якістю надання послуг, модуль обробки запитів та модуль системи управління базами даних. Було визначено, що безпека є ключовою частиною будь-якого процесу розробки програмного забезпечення, тому даний модуль використовує передові стандарти шифрування даних та аутентифікації для того, щоб відповідати підвищеним стандартам безпеки критичної IT-інфраструктури.

Було також розроблено структурну схему для сервера візуалізації даних, що складається з декількох модулів, таких як модуль обробки запитів, модуль системи управління базами даних та модулів агрегації і візуалізації даних. А для візуального представлення цих даних використовується користувацький веб-інтерфейс.

Для розробки компонентів системи було використано підхід по управлінню розробкою програмного забезпечення під назвою Agile. Використання такого підходу дозволило зменшити час на впровадження нових змін в продукт, що забезпечило гнучкість у впровадженні нових змін та водночас підтримувало постійний темп розробки.

Для розгортання системи на серверах підприємства було розроблено схему розгортання компонентів і було проведено розгортання основних компонентів системи – ELK Stack та модулю системи управління якістю надання послуг критичними IT-інфраструктурами.

3 РОЗРОБКА АЛГОРИТМУ РОБОТИ МОДУЛЮ УПРАВЛІННЯ ЯКІСТЮ НАДАННЯ ПОСЛУГ СИСТЕМИ УПРАВЛІННЯ ЯКІСТЮ НАДАННЯ ПОСЛУГ КРИТИЧНИМИ ІТ-ІНФРАСТРУКТУРАМИ

3.1 Розробка алгоритму управління якістю надання послуг критичною ІТ-інфраструктурою

Алгоритм управління якістю надання послуг критичною ІТ-інфраструктурою складається з фундаментальних кроків, в основі яких лежить обробка даних на льоту. Для досягнення поставленої цілі активно використовується модуль системи управління базами даних для отримання даних та основний цикл програми, в ході виконання якого свіжий набір даних потрапляє після попередньої обробки до моделі машинного навчання, яка намагається передбачити стан системи в майбутньому і вплинути на стан за потреби.

Алгоритм починає роботу зі створення запиту до бази даних через модуль системи управління базами даних. В запиті є інформація про версію програмного забезпечення, на основі якої визначається версія моделі машинного навчання, яка є актуальною та сумісною з версією. Як відповідь на запит модуль отримує інформацію про модель та найголовніше – ваги моделі. Ваги моделі це набір матриць – внутрішній стан – моделі машинного навчання, які визначаються під час процесу навчання моделі.

В базі модель може зберігатися в форматі, що є зручним для розробників та обговорюється заздалегідь. Це важливо умова, адже під час розробки важливо узгодити формати, щоб після навчання модель зберігалася в тому ж форматі, що і під час обробки даних. Як показано на рисунку 3.1, після отримання вагів ініціалізується модель машинного навчання. Зазвичай такий процес ініціалізації відбувається наступним чином. В пам’ять сервера завантажується з диска код, що описує архітектуру моделі, а паралельно завантажуються ваги, які використовуються в подальшому.

Рисунок 3.1 – Алгоритм роботи системи управління якістю надання послуг критичними IT-інфраструктурами, перша частина

Наступним етапом роботи алгоритму є цикл, виконання якого триває доти, доки система не отримає команду зупинки. В цьому циклі відбуваються основні події, пов’язані з управлінням. Варто зазначити, що оскільки система є масштабованою, виконання такого циклу може відбуватися паралельно в декількох процесах, що пришвидшує процес управління якістю надання послуг.

В рамках циклу з кожною новою ітерацією в базу даних надсилається запит про отримання наступного набору даних. Якщо система працює в режимі реального часу, то розмір такого набору даних невеликий, але якщо система увімкнена в режимі обробки даних наборами по черзі, то в такому разі час отримання відповіді не відіграє особливої ролі, і система здатна за раз обробляти більші об’єми даних. Тому кількість даних, що отримуються з бази даних на цьому кроці залежить від режиму роботи системи.

Наступним кроком йде попередня обробка даних – це підпроцес системи, адже він включає в себе декілька процесів з обробки даних. Першим ділом дані, які було щойно отримано з бази даних потрібно привести до єдиного вигляду через те, що не всі дані, які потрапляють в базу даних проходять через модуль попередньої обробки даних. Таким чином в базі даних зберігається інформація з різноманітним виглядом та кодуванням, що робить майже неможливим коректну обробку таких даних.

Після приведення даних до єдиного вигляду їх потрібно очистити від даних, які не варто включати до вибірки. В першу чергу це системні логи та трасування, в яких міститься лише інформація у вигляді тексту, та вони не дають ніякої інформативності стосовно стану систему у кількісному виді. Найважливіші дані в такому випадку – це значення кількісних метрик. Найпростіше зробити таку фільтрацію на етапі витягання даних з бази, але через відсутність конкретної схеми даних це інколи буває нелегко, тому на цьому кроці і вирішується дана проблема. Також потрібно позбутися даних, які лежать на одній часовій точці та мають однаковий вигляд – такі дані просто дублюють інформацію та вони не є корисними.

Наступний крок з процесу попередньої обробки даних – це визначення відсутніх значень, що має на меті відновити всі дані, які недостає даному набору даних. Таке інколи трапляється, коли через відсутню схему дані зберігаються в довільному вигляді з недостачею деяких полів. Для виконання такої задачі застосовуються такі прості методи, як заповнення відсутніх даних нулями, середніми або медіанними значеннями. Проте існують підходи, які на основі машинного навчання відновлюють відсутні дані.

Попередньо оброблені дані у вигляді багатомірних тензорів надходять на вхід до моделі машинного навчання, яка зберігається в оперативній пам’яті сервера, як показано на рисунку 3.2. Над вхідним тензором виконуються алгебраїчні операції в графі операцій нейронної мережі з використанням завантажених раніше вагів таким чином, що на вихід з моделі надходить вектор стандартного вигляду, що містить інформацію про стан системи та необхідні операції, які потрібно здійснити для усунення проблем. Такі передбачення робляться для кожного вузла системи.

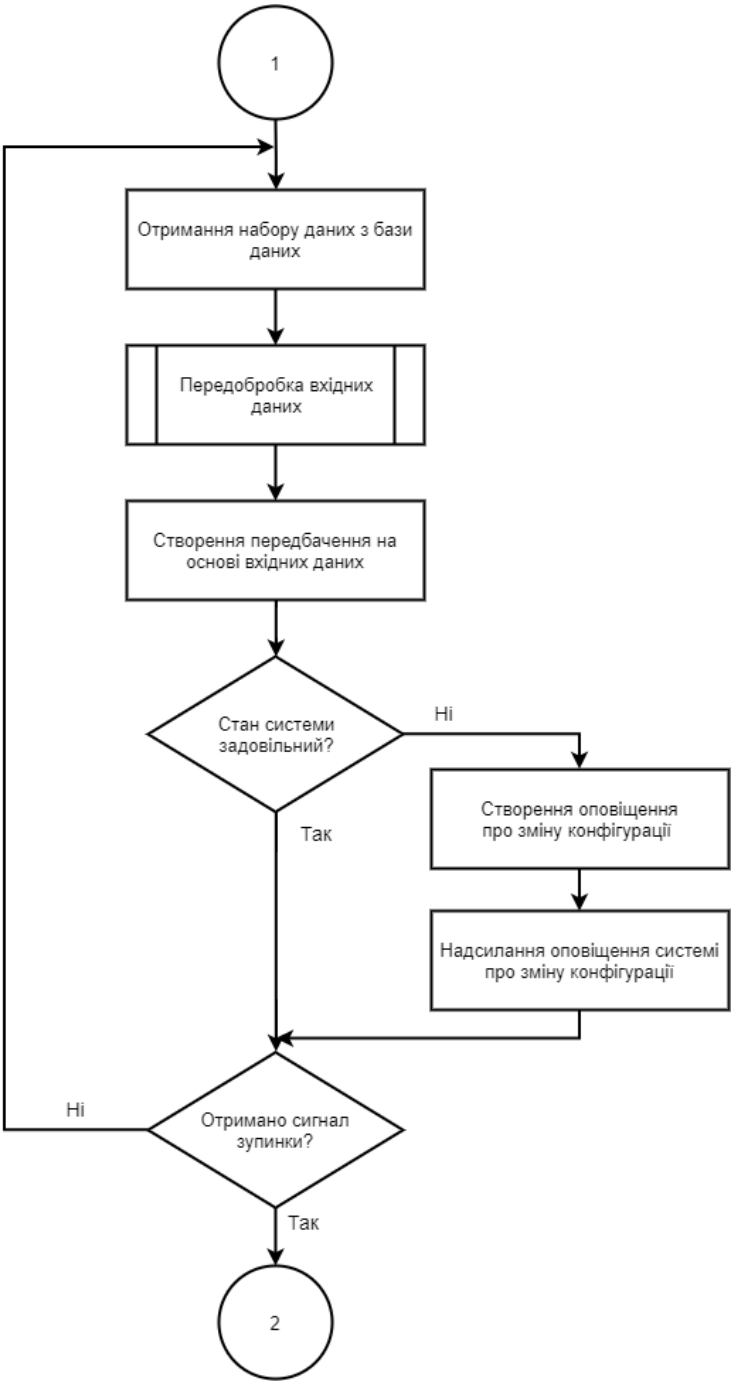

Рисунок 3.2 – Алгоритм роботи системи управління якістю надання послуг критичними IT-інфраструктурами, друга частина

Після цього відбувається перевірка на те, чи результат передбачає можливу помилку у майбутньому, чи ні. Якщо стан задовільний і модель не передбачає можливості помилки, то система працює дані. Якщо ж стан системи виявився незадовільним, то створюється оповіщення системи про потребу змінити

конфігурацію. Вихідний вектор з моделі розшифровується і поля, які відповідають за стан ресурсів системи визначають кількість ресурсів, які потрібно надати або забрати в системи для покращення її рівня надання послуг, як показано на рисунку 3.3.

Рисунок 3.3 – Алгоритм роботи системи управління якістю надання послуг критичними IT-інфраструктурами, третя частина

Оскільки система працює за принципом підписник-видавець, то всі сервери критичної IT-інфраструктури, які по факту є підписниками, отримують від видавця – системи управління якістю надання послуг критичними IT-інфраструктурами – інструкції щодо зміни своїх конфігурацій, якщо в цьому є потреба. Адміністратор сам може налаштовувати сервери після отримання повідомлення вручну, або за нього це може зробити автоматично спеціалізоване програмне забезпечення.

В будь-якому разі відбувається перевірка на те, чи не отримала система сигнал про зупинку. В разі негативної перевірки цикл продовжується – новий набір даних завантажується з бази даних. В разі позитивної перевірки цикл переривається і алгоритм продовжується, як показано на рисунку 3.3. Створюється звіт про виконану роботу для того, аби адміністратору було простіше аналізувати історію роботи системи управління якістю надання послуг.

Даний звіт надсилається на збереження в базу даних через модуль системи управління базами даних. Адміністратор може проглянути історію в майбутньому для виявлення закономірностей в роботі системи і можливої зміни попередніх конфігурацій серверів критичної IT-інфраструктури в майбутньому. По закінченню роботи модулю відбувається звільнення ресурсів, зарезервованих для виконання задач по передбаченню стану системи. Наприклад, моделі машинного навчання вивантажуються з пам’яті, а відкриті підключення до бази даних закриваються для того, щоб уникнути витоку пам’яті.

3.2 Аналіз та вибір інструментів для розробки модулю

Як було згадано в попередніх розділах, одним з найважливіших компонентів будь-якої системи управління якістю надання послуг критичними IT-інфраструктурами, повинна бути система, яка спеціалізується на трьох ключових речах. По-перше, це збирання даних, попередня їх обробка, агрегація і приведення до єдиного вигляду. По-друге, це сховище даних, яке з одного боку підтримує можливість масштабування та реплікації, а з іншого відповідає високим стандартам кібербезпеки. Та, по-третє, це зручна система аналізу зібраних даних. Програмне забезпечення ELK Stack відповідає всім висунутим вимогам, а його функціонал за потреби може бути розширений додатковим програмним забезпеченням.

Ще одним з ключових компонентів системи є алгоритм, який здатен знаходити в даних закономірності, запам’ятовувати їх та відтворювати для знаходження відповіді на питання. Такі алгоритми дістали назву алгоритмів машинного навчання.

Алгоритми машинного навчання використовують статистику для пошуку шаблонів у великих масивах даних. Варто зазначити, що дані можуть бути будь-якого вигляду – цифрові дані, текстові дані, зображення, відео, логи та події тощо. Будь-яка інформація, якщо вона може бути збережена цифровим шляхом, може подаватися на вхід до алгоритмів машинного навчання.

Використання машинного навчання зумовило успіх багатьох сервісів, якими люди користуються щодня – рекомендаційні системи, такі як Netflix, YouTube та Spotify; такі пошукові системи, як Google і Baidu; канали соціальних медіа як Facebook та Twitter; голосові помічники, додатки для оброблення фото, відео, комп'ютерні ігри, і це ще не повний список.

В кожному з таких сервісів відбувається декілька речей. По-перше, всі ці платформи намагаються зібрати якомога більше даних про вас – які жанри музики користувач полюбляє слухати, які фільми любить дивитися, які посилання користувач натискає, на які статуси реагує яким чином – і за допомогою машинного навчання, такі компанії намагаються зробити здогадку про те, що користувачу сподобалось би наступного разу.

Насправді, цей процес досить простий та прямолінійний: алгоритм шукає певну закономірність, а потім використовує її для створення передбачень. Але навіть така проста схема в значній мірі керує світом сьогодення. А історія машинного навчання почалася завдяки винаходу, який розробив в 1986 році Джеффрі Хінтон, сьогодні відомий як батько глибокого навчання. Глибоке навчання – це сучасна назва машинного навчання, застосованого до великих масивів даних, що використовує техніку, яка дає комп'ютерам додаткову здатність знаходити, виокремлювати та посилювати навіть найменші закономірності, знайдені в даних. Інструмент, що зазвичай використовується в глибокому навчанні називається глибокою нейронною мережею. Нейронна мережа такого типу називається глибокою, оскільки вона має дуже багато шарів лінійних обчислювальних вузлів, які працюють об'єднано, щоб збирати дані та отримувати кінцевий результат у вигляді передбачення.

3.2.1 Аналіз різновидів моделей машинного навчання

Не менш важливо необхідно зазначити, що машинне навчання у будь-якому своєму прояві поділяється на два основних види: навчання з учителем та навчання без учителя. Так у задачах машинного навчання з учителем, найбільш розповсюдженому виду машинного навчання, дані позначаються міткою або назвою, щоб надати можливість машині зрозуміти, які саме закономірності слід шукати. Наприклад, коли користувач натискає клавішу програвання на сервісі Netflix – користувач тим самим говорить алгоритму, які шоу йому подобаються, щоб алгоритм зміг шукати подібні шоу в майбутньому. Представник такого виду машинного навчання – нейронна мережа, зображена на рисунку 3.4.

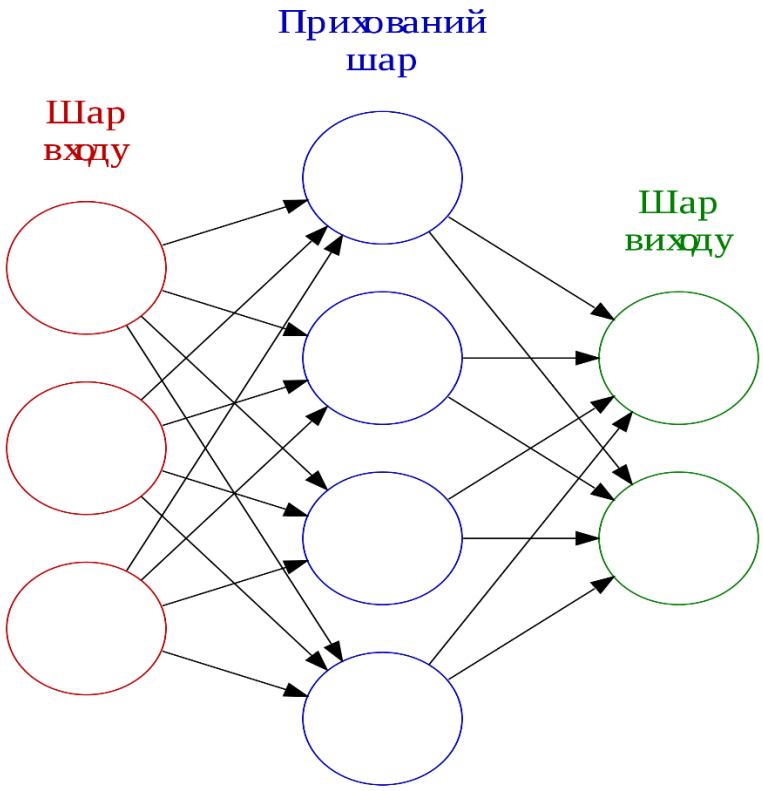

Рисунок 3.4 – Схематичне зображення штучної нейронної мережі

Другий вид машинного навчання – машинне навчання без учителя. У навчанні без учителя дані не мають міток, вони не марковані. Комп’ютер просто шукає будь-

які закономірності, які він зможе знайти. Такий вид машинного навчання є не настільки популярним, оскільки результат роботи доволі складно перевірити. Цікаво, що такі підходи стали дуже популярними в кібербезпеці.

Кількість задач, в яких використовуються нейронні мережі, зростає кожного дня, адже ці алгоритми зарекомендували себе як прекрасна та більш дешева заміна інших алгоритмів та евристик, в першу чергу через те, що їх дешевше впроваджувати і простіше переналаштовувати. Через таку велику кількість доменів, в яких нейронні мережі знайшли собі застосування, науковці постійно створюють такі архітектури нейронних мереж, які б краще виконували ту чи іншу задачу. Наприклад, звичайна нейронна мережа зазвичай складається з вхідного шару нейронів, вихідного та декількох прихованих шарів.

Проте існують і більш складні нейронні мережі, здатні обробляти особливі типи даних. Так згорткові нейронні мережі можуть якісно та швидко обробляти дво- та тривимірні зображення. Генеративні нейронні мережі здатні на основі двовимірних зображень генерувати нові зображення. А рекурентні нейронні мережі знайшли застосування в задачах обробки історичних даних.

Принцип роботи рекурентних нейронних мереж схожий на принцип мислення людини. Люди не починають своє мислення з нуля щосекунди. Наприклад, читаючи книжку, людина розуміє кожне слово, ґрунтуючись на розумінні попередніх слів. Також людина не кидає усе і не починає думати з нуля кожного разу. Думки людини мають певну постійність. Рекурентні нейронні мережі, як показано на рисунку 3.5, – чудовий приклад штучних нейронних мереж, які мають схожі властивості.

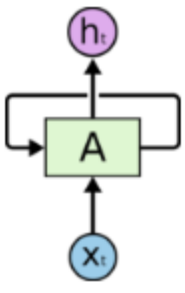

Рисунок 3.5 – Схематичне зображення рекурентної нейронної мережі

Але традиційні нейронні мережі не пристосовані до запам’ятовування інформації і це булло головним їх недоліком протягом багатьох років. Наприклад, якщо стоїть задача, що потрібно визначити та класифікувати, яка подія відбувається у кожній точці фільму, то незрозуміло, як звичайна нейронна мережа могла б використовувати свої міркування про попередні події у фільмі для інформування про пізніші події. Рекурентні нейронні мережі дають відповідь на це питання. Такі мережі являють собою мережі з рекурентними зв’язками в них, що дозволяють ефективно зберігати інформацію.

На рисунку 3.5 зображено фрагмент нейронної мережі A, що має деякий вхід x_t і видає значення h_t. Цикл дозволяє передавати інформацію від одного кроку мережі до іншого. Рекурентна нейронна мережа може розглядатися як декілька копій однієї мережі, при чому кожна з попередніх мереж передає повідомлення наступній, як показано на рисунку 3.6.

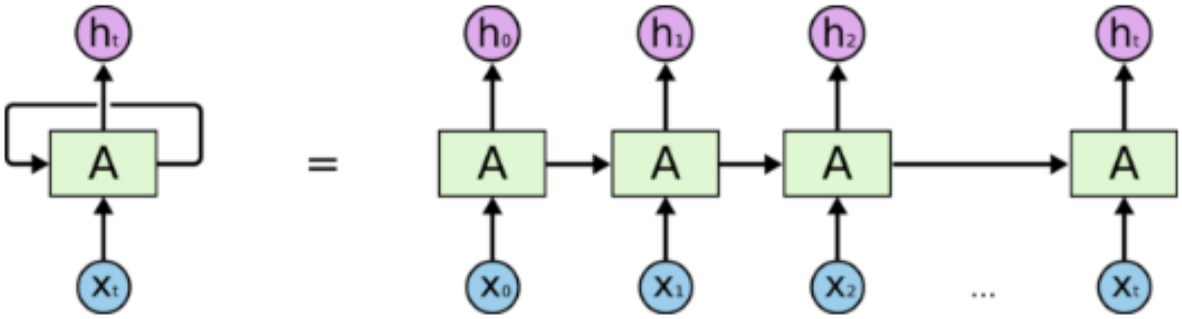

Рисунок 3.6 – Схематичне зображення еквівалентної нейронної мережі до рекурентної нейронної мережі

Якщо спробувати «розкрутити» рекурентну нейронну мережу, як показано на рисунку, можна виявити, що природа рекурентних нейронних мереж тісно пов'язана з поняттями послідовності та списками. Саме така архітектура нейронних мереж використовується для таких даних, які мають послідовну природу. І сьогодні вони використовуються в багатьох задачах. В останні кілька років було досягнуто неймовірного успіху в застосуванні RNN до різноманітних проблем: розпізнавання

мови, моделювання мови, переклад з однієї мови на іншу, знаходження підписів до зображень тощо.

Яскравим прикладом обмеженості звичайних повнозв'язних нейронних мереж (а також згорткових нейронних мереж) є те, що їх прикладний програмний інтерфейс є занадто обмеженим: вони приймають вектор фіксованого розміру на вхід (наприклад, зображення або відео) і віддають вектор фіксованого розміру на вихід (наприклад, ймовірності різних подій). Окрім того ці моделі виконують таке відображення, використовуючи фіксовану кількість обчислювальних кроків (наприклад, кількість шарів у моделі).

Основна причина того, що рекурентні нейронні мережі є більш цікавими в нашому випадку, полягає в тому, що такі алгоритми дозволяють працювати над послідовностями даних. Наприклад, можна використовувати рекурентні мережі для отримання послідовності на вході, виході з мережі або в загальному випадку і те, і інше, як показано на рисунку 3.7.

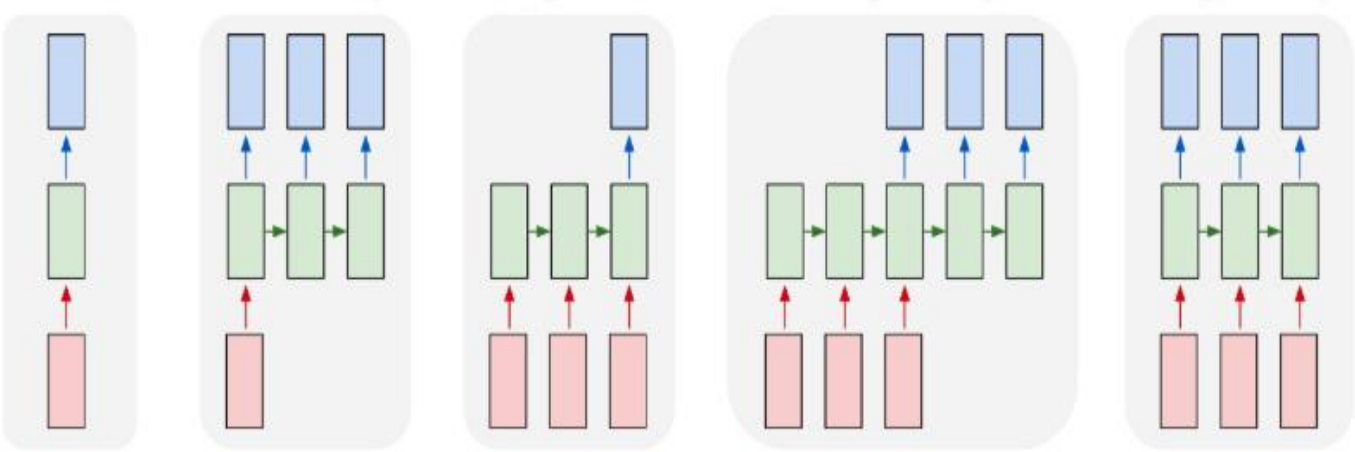

Рисунок 3.7 – Приклад комбінацій вхідних та вихідних даних в різних видах нейронних мереж

На даному рисунку схематично кожен прямокутник представляє вектор, а стрілки представляють функції (наприклад, множення матриці). Вхідні вектори червоного кольору, вихідні вектори синього та зеленого кольорів зберігають в собі

стан рекурентної нейронної мережі. Як показано на рисунку, такі нейронні мережі працюють у декількох режимах:

- звичайний режим роботи без рекурентності, на вхід подається вектор фіксованого розміру, на виході отримується вектор також фіксованого розміру. Прикладом застосування таких мереж можуть бути задача класифікації зображень;
- режим роботи з послідовним виводом. Наприклад, в задачах знаходження підпису до зображення саме зображення приймають як вхід та виводять речення з слів;
- режим роботи з послідовним введенням даних. Наприклад, аналіз сентиментів, коли певне речення класифікується як вираження позитивних чи негативних настроїв;
- режим роботи з послідовним введення даних на вхід та виведенням результату у вигляді послідовності. Наприклад, у задачах машинного перекладу, RNN читає речення англійською мовою, а потім виводить речення українською мовою;
- режим роботи з синхронізованою послідовність введення та виведення. Як приклад можуть слугувати задачі класифікації відео, де потрібно позначити кожен кадр відео.

Принцип роботи алгоритму зображений на рисунку 3.8.

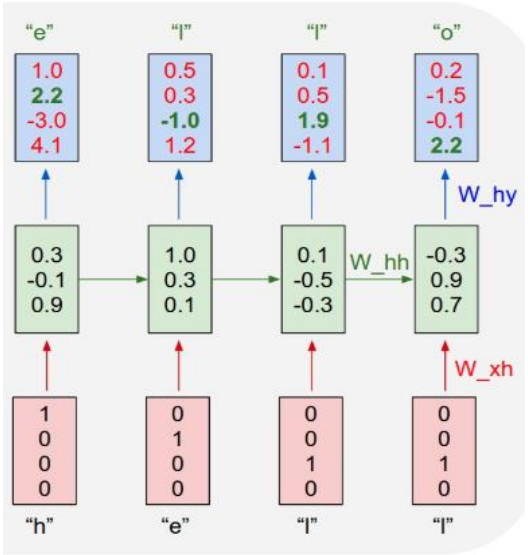

Рисунок 3.8 – Приклад передачі параметрів в середині рекурентної нейронної мережі

Принцип роботи алгоритму полягає в тому, що на вхід нейронна мережа отримує вектор з даними x та певний вектор y отримує на вихід, який протиставляється вхідному вектору. На рисунку 3.8 зображено приклад передачі параметрів в середині рекурентної нейронної мережі параметри W_hh – це матриця з параметрами, яка базується на попередньому прихованому стані, W_xh – це матриця з параметрами, яка базується на поточному вхідному векторі і W_hy – це матриця з параметрами, яка базується на вихідному значенні.

Для розрахунку поточного внутрішнього стану нейронної мережі використовується формула 3.1:

$$h_t = f(h_{t-1}, x_t), \tag{3.1}$$

де h_t – це поточний стан системи, h_{t-1} – це попередній стан системи, а x_t – це значення параметрів, що подаються на вхід. Для розрахунку значення поточного стану систему зазвичай використовуються функцію гіперболічного тангенса, а загалом функція має вигляд, як показано в формулі 3.2:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t), \tag{3.2}$$

де W_{hh} – це матриця вагів на рекурентному шарі, а W_{xh} – це значення матриці вагів на вхідному шарі нейронної мережі. Щоби розрахувати значення вихідного вектора застосовується формула 3.3:

$$y_t = W_{hy}h_t, \tag{3.3}$$

де y_t – це значення вихідного вектора, а W_{hy} – це значення матриці вагів на вихідному шарі нейронної мережі. Загалом процес навчання такого алгоритму зводиться до наступних кроків:

- перший набір даних надсилається в нейронну мережу;

- на основі значення попереднього стану мережі та поточного вхідного вектора розраховується поточний прихований стан мережі;
- значення поточного прихованого стану стає попереднім станом мережі для наступної порції даних;
- процес повторюється доки всі дані не будуть пропущені через рекурентну нейронну мережу;
- одразу після цього кінцевий поточний стан мережі використовується, щоб розрахувати результуючий вектор;
- результуючий вектор порівнюють з очікуваним результатом, та розраховують значення помилки;
- значення помилки потім поширюється назад в мережі, а ваги мережі змінюються в залежності від значення помилки.

Після закінчення процесу тренування нейронної мережі ваги з матриці вагів на шарах нейронної мережі можуть бути збережені. Для того, щоб отримати передбачення наступного разу ваги потрібно завантажити в мережу та використати операцію проходу вперед, під час якої вхідний вектор направляється на вхід до нейронної мережі, де над даним вектором виконуються лінійні алгебраїчні операції, та отримується вихідний вектор.

3.2.2 Вибір архітектури штучної нейронної мережі

Оскільки для рішення поставленої задачі було вирішено використовувати рекурентні нейронні мережі, дуже важливо вибрати правильну архітектуру мережі, щоб отримати найкращі результати за мінімальний час. Звичайні рекурентні нейронні мережі страждають від проблеми короткочасної пам'яті. Якщо послідовність даних достатньо довга, їм стає важко переносити інформацію від попередніх часових кроків до пізніших. Тож на довгих послідовностях даних, якщо треба зробити передбачення, RNN може залишити важливу інформацію, яка була виявлена спочатку.

Під час зворотного розповсюдження помилки рекурентні нейронні мережі страждають від проблеми зникаючого градієнтну. Градієнти в даному – це значення,

які використовуються для оновлення вагів нейронної мережі. Проблема градієнтів, що зникає, полягає в тому, що значення градієнтів сильно зменшується, оскільки вони поширюються назад через попередні кроки. Якщо значення градієнта стає вкрай малим, швидкість навчання сильно падає. Таким чином у рекурентних нейронних мережах шари, які отримують невелике градієнтне оновлення, перестають навчатися. Зазвичай це більш ранні шари. Оскільки ці шари не вчаться, RNN може забути побачену в більш довгих послідовностях інформацію, тим самим маючи короткочасну пам'ять.

Алгоритми LSTM та GRU були створені як рішення для проблеми короткочасної пам'яті в рекурентних нейронних мережах. Вони мають внутрішні механізми, які називаються воротами, які можуть регулювати потік інформації. Такі ворота можуть визначати те, які дані в послідовності важливо зберегти або викинути. Роблячи це, можна передавати відповідну інформацію по довгому ланцюгу послідовностей, щоб робити передбачення. Практично всі сучасні програми, засновані на використанні рекурентних нейронних мереж, досягають результатів за допомогою використання цих двох типів мереж.

LSTM має аналогічний спосіб управління потоком даних, як і рекурентна нейронна мережа. Даний алгоритм обробляє дані, що передають інформацію, по мірі їх поширення вперед. Відмінності полягають у внутрішній будові нейронів LSTM. Така будова дозволяє використовувати LSTM для зберігання або забування інформацію. Провідна концепція LSTM – це збереження стану комірок, і так звані ворота. Стан комірки виступає як транспортна магістраль, яка передає інформацію про навчання моделі як ланцюгу послідовностей. Це можна назвати "пам'яттю" мережі. Стан комірки, теоретично, може нести відповідну інформацію протягом усього процесу обробки послідовності. Тож навіть інформація з попередніх етапів часу може впливати на результат пізніших часових кроків, зменшивши ефекти короткочасної пам'яті.

Коли стан комірки передається далі, інформація додається або видаляється зі стану комірки за допомогою використання воріт. Ворота – це невеликі прототипи

нейронних мереж, які вирішують, яку інформацію додавати або видаляти зі стану комірок. Ворота містять сигмоїдні функції активації. Сигмоїдна функція активація схожа на активацію гіперболічного тангенса. Замість того, щоб заганяти значення в рамки між –1 і 1, така функція заганяє значення в рамки між 0 і 1. Це корисно для оновлення або забуття даних, оскільки будь-яке число, помножене на 0, дорівнює 0, призводячи до того, що деякі значення будуть залишатися або забуватися. Разом з тим, будь-яке число, помножене на 1, є тим самим значенням, тому значення залишається однаковим або зберігається. Мережа може дізнатися, які дані не важливі, і їх можна забути, або які дані важливі, і їх потрібно зберігати. Такий підхід називається гратами забування.

Але окрім такого виду нейронних мереж, як LSTM, існують мережі типу GRU – це нове покоління рекурентних нейронних мереж, яке чимось подібне до вищезгаданих LSTM. Алгоритм GRU не використовує підхід використання стану самої комірки та для передачі інформації використовує прихований стан самої мережу. Цей алгоритм також використовує двоє граток: грати скидання та грати оновлення стану. Грати оновлення діють схоже на те, як працюють ворота забуття в LSTM. Цей компонент, можна сказати, вирішує, яку інформацію забути та видалити, а яку нову інформацію додати. Грати скидання – це ще один тип граток, який використовується для визначення попередньої кількості інформації, яку потрібно забути.

Підсумовуючи вище сказане, у алгоритмі машинного навчання GRU менше тензорних операцій, отже, такий алгоритм трохи швидше тренується за алгоритм LSTM. Оскільки система має обмежений набір ресурсів, то було вирішено використовувати алгоритм машинного навчання GRU, що пришвидшило як процес навчання моделі, так і процес передбачення результатів.

3.3 Розробка програмного забезпечення підсистеми управління якістю надання послуг

Першим кроком розробки програмного забезпечення підсистеми управління якістю надання послуг критичною IT-інфраструктурою підприємства стала розробка схеми класів та визначення основних об’єктів підсистеми. Основною ідеєю даного додатку стало те, що всі основні функції контролюються і виконуються одним основним класом, що має назву SystemQualityManager. Даний клас, як показано на рисунку 3.9, інкапсулює в собі інформацію про з’єднання до бази даних та кешу, чергу повідомлень, алгоритм тренування нейронної мережі і алгоритм управління якістю надання послуг критичною IT-інфраструктурою.

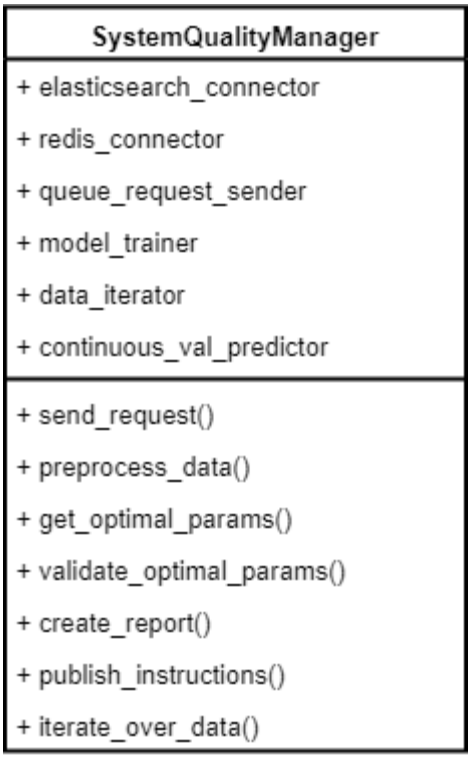

Рисунок 3.9 – Схема класу SystemQualityManager

Використовуючи принцип об’єктно-орієнтованого програмнування інкапсуляція, даний клас включає в себе поля, які відповідають за внутрішній стан об’єкту даного класу. Поля мають різні типи, але оскільки програмне забезпечення розроблене на мові Python, що має динамічну типізацію, то не обов’язково вказувати

тип змінних в класі. На початку роботи об’єкта класу його поля ініціалізуються об’єктами відповідних класів, наприклад, поле, що відповідає за зберігання посилання на об’єкт класу, що регулює з’єднання з базою даних, називається elasticsearch_connector. Варто зазначити, що така назва обумовлена використанням спеціальної нотації при розробці на даній мові програмування – зміїна нотація, яка регламентує те, що всі назви полів та внутрішніх методів класу повинні мати назву, яка починається з малої букви, а слова в назві поділені за допомогою нижнього прочерку. Така нотація встановлена стандартами мови програмування та забезпечує дотримання коду чистим та таким, що легко читається.

Окрім полів об’єкт даного класу містить також додаткові методи, такі як send_request() – метод, що відповідає за створення запиту до бази даних, очікування відповіді від бази даних і приведення даних до вигляду, який сприймається інтерпретатором мови програмування. В разі виникнення помилок, даний метод коректно їх оброблює, наприклад, якщо з’єднання до бази даних не було встановлено протягом певного проміжку часу, то буде надіслано повторний запит на отримання даних, а в разі повторної помилки з’єднання користувача буде оповіщено про наявність помилки.

Метод, що називається preprocess_data() відповідає за попередню обробку даних, що включає в себе приведення даних до єдиного вигляду, фільтрація неінформативних даних, заповнення відсутніх даних тощо. За допомогою методу get_optimal_params() модель машинного навчання тренується на нових даних і якщо результат моделі покращився, дані помічаються як оптимальні та записуються в базу даних. Даний метод використовує алгоритм машинного навчання, який приведений у додатку В. Після отримання оптимальних параметрів процеси, які використовують попередні версії моделей машинного навчання отримують повідомлення про те, що існує можливість оновити параметри моделей. Якщо в процесах моделі знаходяться в стані простою, відбувається оновлення параметрів. В разі, якщо оптимальних параметрів не знайдено або немає нових тренувальних даних, повертаються оптимальні параметри з попередньої ітерації.

Для того, щоб зрозуміти, чи покращилася якість моделі, використовується метод validate_optimal_params(), який інкапсулює різні метрики валідації моделі машинного навчання і повертає результат роботи – позитивна відповідь в разі, коли якість моделі було покращено та негативний в інших випадках. В разі отримання сигналу про зупинку, викликається метод create_report(), який створює підсумковий аналіз роботи, яку було зроблено та зберігає ці дані в базу даних для можливого подальшого аналізу роботи системи адміністратором. Метод publish_instructions() використовує чергу повідомлень для того, щоб надсилати повідомлення про зміну конфігурації системи за потреби критичній IT-інфраструктурі. А за допомогою методу iterate_over_data() створюється ітератор, який використовується для проходження по даним в циклі. Це робиться для того, аби дані не оброблялися одразу великими кількостями, а динамічно генерувалися в циклі.

Інтерфейс DatabaseConnector є стандартним для всіх класів, які абстрагують підключення до бази даних та надає користувачу єдиний набір методів та полів для підключення до бази даних та виконання запитів. Два класи наслідують цей інтерфейс: RedisConnector та ElasticsearchConnector, кожен з яких відповідає за підключення до одного з двох типів баз даних, що використовуються програмним забезпеченням.

Клас ElasticsearchConnector, як показано на рисунку 3.10, відповідає за підключення до бази даних Elasticsearch, яка є центральним сховищем логів та метрик системи. Клас реалізовано, як обгортку над стандартною бібліотекою для доступу до бази даних Elasticsearch та підтримується розробником самої бази даних. Даний клас надає можливість отримати підключення до бази даних, виконати запит до бази або декілька запитів одночасно. Результат, який повертається залежить від вигляду запитів, наприклад, якщо запит на додавання інформації в базу, то та ж сама інформація має повернутися на вихід з методу, якщо не відбулася помилка; якщо ж запит був на пошук даних, то на вихід з методу буде повернуто об’єкти, які відповідають запиту або порожня відповідь, якщо таких не було знайдено.

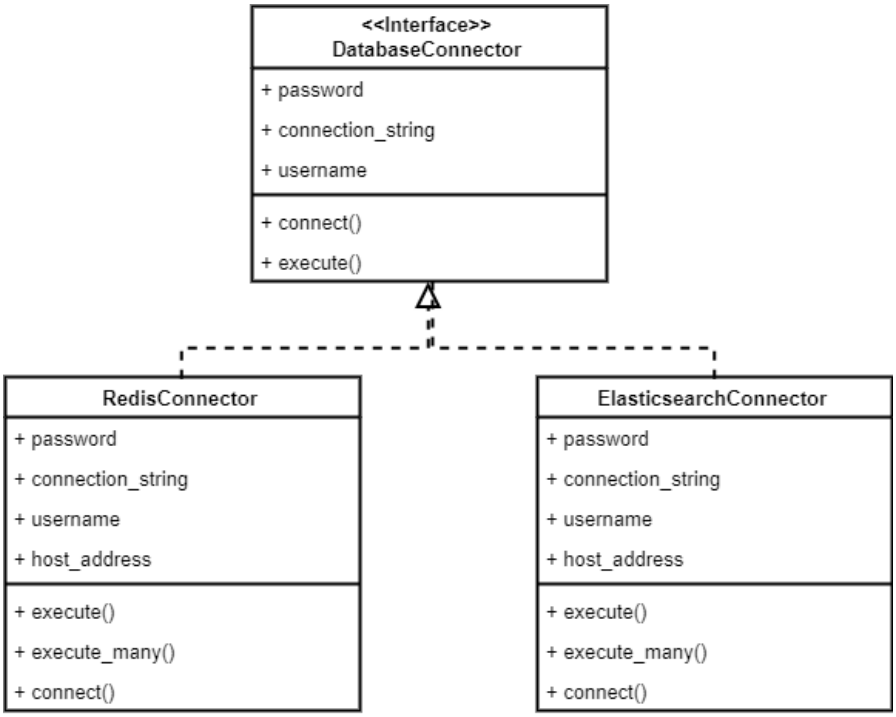

Рисунок 3.10 – Схема класів підключення до бази даних

Клас RedisConnector дозволяє налаштовувати з’єднання до бази даних Redis, яку можна використовувати як сховище для кешу даних. Такий підхід кешування дозволяє збільшити швидкість обробки даних через зберігання деякої інформації в оперативній пам’яті, що дозволяє отримувати до неї доступ майже одночасно з надісланим запитом. Даний клас також реалізовує всі методи інтерфейса і інкапсулює поля, в яких зберігаються дані для підключення – логін, пароль, адреса сервера з базою даних тощо.

ModelScorer являє собою інтерфейс з методами валідації моделей машинного навчання, як показано на рисунку 3.11. Даний підхід розроблено з використанням шаблону стратегія. Стратегія являє собою шаблон проектування, який визначає набір алгоритмів, інкапсулює кожен з них і забезпечує їх взаємозамінність. Залежно від ситуації можна легко замінити один використовуваний алгоритм іншим. При цьому заміна алгоритму відбувається незалежно від об'єкта, який використовує даний алгоритм. Тому в даному випадку через наявність декількох стратегій, щодо валідації моделі з використанням різних метрик було доцільно використати такий шаблон проектування.

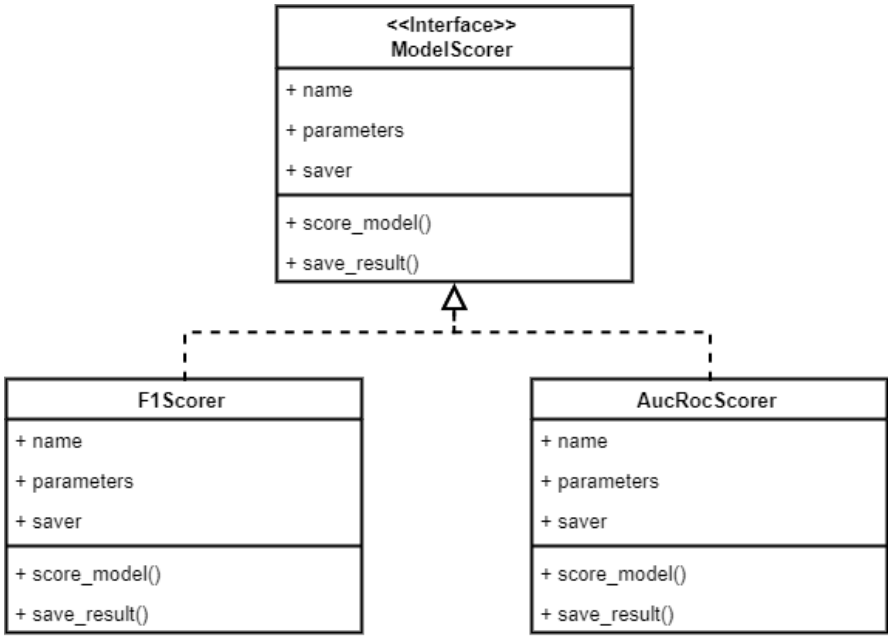

Рисунок 3.11 – Класи валідації моделей машинного навчання

Кожен клас, в залежності від обраної стратегії, виконує оцінку моделі машинного навчання. Так, клас F1Scorer імплементує всі методи свого інтерфейсу. Метод score_model() використовується даним класом для оцінювання моделі машинного навчання за допомого оцінки f1. Результати оцінювання повертаються назад з методу, а результат оцінювання може бути збереженим. У даного класу є поля, які також наслідуються від спільного інтерфейсу та слугують в першу чергу для коректної роботи шаблону проектування стратегія. Наприклад, поле name дає інформацію про назву стратегії оцінки моделі, а поле parameters зберігає інформацію про параметри моделі, з якими вона була натренована.

Аналогічно, клас AucRocScorer наслідується від інтерфейсу ModelScorer та забезпечує оцінку моделей машинного навчання, але з використанням алгоритму AUC ROC. Даний алгоритм дозволяє всебічно дослідити якість моделі та показати кількісну характеристику якості. В разі потреби результат оцінювання можна записати в базу даних.

Клас QueueRequestSender, який зображено на рисунку 3.12, має доволі простий інтерфейс – один метод та одне поле. Метод send_message() слугує абстракцією над

механізмом надсилання запитів до черги повідомлень, яка використовується в першу чергу для того, щоб опубліковувати повідомлення про зміну конфігурації системи. Для підключення до існуючої черги повідомлень об’єкт класу має ініціалізоване поле connection_string, в якому зберігається строка підключення. Використання черги повідомлень дозволяє серверу працювати в асинхронному режимі без потреби очікувати на відповідь від сервера.

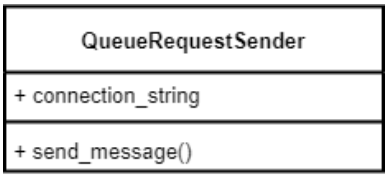

Рисунок 3.12 – Клас надсилання запитів до черги повідомлень

Для ефективного тренування моделей машинного навчання існує єдиний клас – ModelTrainer, як показано на рисунку 3.13, принцип роботи якого описано в додатку В. Даний клас відповідає за поділ даних на тренувальний та тестувальний набори даних, виконання процесу тренування моделі машинного навчання, валідацію результатів, а за наявності покращення оцінки, клас може зберігати оновлені параметри моделі в базу даних.

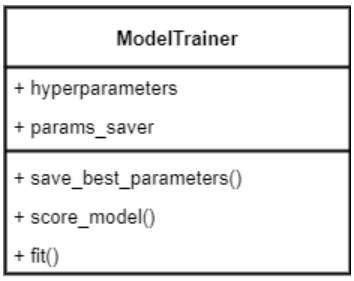

Рисунок 3.13 – Клас тренування моделі машинного навчання

Було також розроблено абстрактний клас StatePredictor, як показано на рисунку 3.14, який використовується для створення абстракції над моделями машинного навчання різних типів.

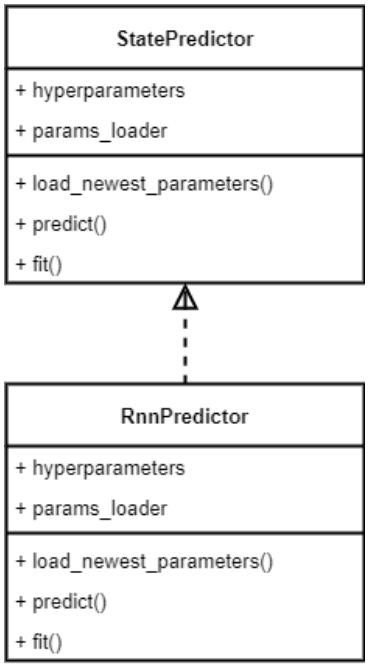

Рисунок 3.14 – Класи моделей машинного навчання

Основний функціонал класу забезпечується трьома методами, кожен з яких відповідає за певну функцію. Метод load_newest_parameters() дозволяє завантажувати найбільш актуальні параметри моделі задля того, щоб ініціалізувати модель після створення екземпляру класу. Метод fit() – це стандартна назва для всіх методів, які використовуються для того, щоб надсилати дані в модель машинного навчання та тренувати її. Для того, щоб отримати результат передбачення на нових даних у випадку валідації моделі або у випадку стандартного режиму роботи, використовується метод predict(), назва якого також є стандартною для моделей машинного навчання. RnnPredictor – це конкретний клас, який наслідується від базового класу та реалізовує всі ці методи і поля. В основі даного класу лежить модель рекурентної штучної нейронної мережі.

3.4 Висновки до розділу

В даному розділі було розроблено алгоритм управління якістю надання послуг критичною IT-інфраструктурою підприємства. Для досягнення поставленої цілі активно використовується модуль системи управління базами даних для отримання

даних та основний цикл програми, в ході виконання якого новий набір даних потрапляє після попередньої обробки до моделі машинного навчання, яка намагається передбачити стан системи в майбутньому і вплинути на стан за потреби. В разі виникнення підозри на проблему створюється системне оповіщення, яке надсилається на сервери критичної IT-інфраструктури.

Було обрано модель машинного навчання, яка дає змогу найкраще вирішити задачу предиктивного моніторингу. Було порівняно різні види сучасних архітектур моделей штучних рекурентних нейронних мереж та було обрано таку, яка найкраще відповідатиме поставленим задачам. Таким алгоритмом було обрано рекурентні нейронні мережі на основі GRU.

Як перший етап розробки програмного забезпечення підсистеми управління якістю надання послуг критичними IT-інфраструктурами, було розроблено схему класів даного програмного забезпечення. В основі програми було використано ідею про те, що всі основні функції повинні виконуватися одним основним класом, який дістав назву SystemQualityManager. Даний клас інкапсулює в собі інформацію про з’єднання до бази даних та кешу, чергу повідомлень, алгоритм тренування нейронної мережі і алгоритм управління якістю надання послуг критичною IT-інфраструктурою.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ УПРАВЛІННЯ ЯКІСТЮ НАДАННЯ ПОСЛУГ КРИТИЧНИМИ ІТ-ІНФРАСТРУКТУРАМИ

4.1 Підготовка до навчання моделі штучної нейронної мережі

Важливим кроком до підготовки моделі машинного навчання є: вибір метрики для перевірки якості моделі, створення набору даних, на яких буде навчатися алгоритм та налаштування гіперпараметрів. Моделі не обов'язково повинні постійно навчатись, щоб потім бути впровадженими в виробництво. Досить часто модель може бути просто одноразово навчена спеціалістом і відправлена на використання на виробництві, поки її продуктивність не погіршиться настільки, щоб виникла потреба її оновити.

Сама по собі модель являє собою набір команд, код, який написаний з використанням спеціальних фреймворків машинного навчання. Але окрім самого коду, який запускає модель, є ще так звані ваги моделі, які використовуються саме для проведення операції передбачення. Такі дані можна зберігати в форматі Pickle, який перетворює об'єкт, створений за допомогою мови Python у бітовий потік і дозволяє його зберігати на диску та перезавантажувати в подальшому. Це хороший та гнучкий формат для зберігання моделей машинного навчання за умови, якщо програми, що виконують цей код також написані на мові Python.

Ще один формат для експорту моделей машинного навчання – ONNX. Це відкритий формат обміну нейронними мережами – відкритий тому, що підтримує зберігання та перенесення прогнозної моделі між фреймворками та мовами. Більшість бібліотек глибокого навчання підтримують цей формат, а бібліотека sklearn також має розширення, які дозволяють перетворення моделей, створених з допомогою такої бібліотеки у формат ONNX.

PMML або мова розмітки для моделей машинного навчання – це ще один формат обміну моделей машинного навчання. Як і для ONNX, sklearn також має розширення, за допомогою якого можна перетворювати моделі у формат PMML. Однак цей формат має недолік, який полягає в тому, що PMML підтримує лише певні

типи моделей машинного навчання. Програма MML існує ще з 1997 року і тому існує великий набір додатків, які використовують цей формат. Такі програми, як, наприклад, SAP, можуть використовувати певні версії стандарту PMML, як і CRM-програми, такі як PEGA.

POJO та MOJO – це експортний формат, що використовується компанією H2O.ai, який має на меті запропонувати модель, яка легко вбудовується у java-додаток. Однак цей стандарт дуже специфічний і використовується виключно платформою H2O, тому стандарт ONNX найбільш підходить для даної задачі.

4.1.1 Поділ даних на тренувальний та тестовий набори

Для ефективного навчання моделей машинного навчання недостатньо просто обрати модель, підібрати правильні параметри та відправляти модель на використання на виробництві. Дуже важливу роль відіграє процес збору та обробки даних. І разом з тим потрібно усвідомлювати, що навіть якщо дані зібрано і модель почала процес тренування, така модель може все одно страждати від двох проявів неправильного навчання моделей: перенавчання та недонавчання.

Перенавчання означає, що модель, яка тренувалася, робила це «надто добре», і тепер така модель надто добре знає дані, на яких тренувалася. Зазвичай це відбувається, коли модель занадто складна, тобто містить занадто багато функцій та змінних, порівняно з кількістю даних. Така модель буде дуже точною на даних, які використовувалися для навчання, але, ймовірно, буде дуже неточною на непідготовлених чи нових даних. Так стається тому, що модель не змогла апроксимувати цільову функцію, тобто результати роботи такої моделі дуже конкретні, тому не можна робити ніяких висновків щодо інших даних. В основному, коли таке відбувається, модель вивчає або описує шум у навчальних даних, замість фактичних зв’язків між змінними в даних. Цей шум, очевидно, не входить до жодного нового набору даних, і не може бути застосований до нього.

На відміну від перенавчання, бувають ситуації, коли модель є не до кінця навченою. Це означає, що модель не відповідає навчальним даним і, таким чином, не

має можливості бачити тенденції та закономірності в даних. Це також означає, що модель не може бути застосована до створення передбачень на нових даних. Очевидно, що в більшості випадків це, як правило, результат навчання дуже простої моделі, для якої було недостатньо прогнозів, незалежних змінних. Також може статися, що, наприклад, застосовуючи лінійну модель таку, як лінійна регресія, до даних, які не є лінійними, можна отримати результат у вигляді недонавченої моделі. Цілком зрозуміло, що така модель матиме слабку здатність до прогнозування на тренувальних даних і не може бути взагалі застосована щодо інших даних.

Один з підходів, що використовується для запобігання перенавчання моделей машинного навчання є підхід поділу даних на два набори: тренувальні дані та дані для тестування. Але звичайний поділ даних не є панацеєю від перенавчання моделі. На практиці дуже часто використовується підхід крос-валідації [15], який дуже схожий на звичайний поділ даних на тренувальну та тестувальну вибірки, але він застосовується до більшої підмножини. Це означає, що дані розділяються на k підмножин та тренуються на k-1 таких підмножин. Далі відбувається процес тестування оцінки моделі на останній підмножині для тесту. Так можна зробити для кожної з підмножин.

Метод крос-валідації, який дістав назву K-Folds дані розділяються на k різних підмножин. В процесі навчання моделі використовується підмножини k-1 для підготовки даних, і залишається остання підмножина у якості тестових даних. Потім значення метрик для моделі порівнюється на кожній зі підмножин, а потім модель доопрацьовується.

Це ще один метод крос-валідації називається LOOCV (залишаючи один об’єкт поза множиною). У цьому типі крос-валідації кількість підмножин дорівнює кількості спостережень, які наявні в наборі даних. Кожна з цих підмножин потім порівнюється окремо, а потім відбувається процес тестування моделі на останній підмножині. Оскільки в такому методі велика кількість навчальних наборів даних, цей метод є дуже обчислювально дорогим і його слід використовувати на невеликих наборах

даних. Якщо набір даних великий, то, швидше за все, краще використовувати інший метод.

Існує багато інших методів крос-валідації даних. Одні з них базуються на використанні попередніх знань про дані, розподілу імовірностей, створенні моделі даних, інші ґрунтуються на використанні евристичних підходів. Загалом, кожен з таких підходів зводиться до принципу поділяй та володарюй. А використання поділу даних, хоч і сповільнює процес навчання моделей машинного навчання, проте надає моделі можливість тренуватися на нових комбінація даних щоразу, що значно зменшує вірогідність того, що модель перенавчиться. Відповідно, принципи поділу даних було використано і при проведенні експерименту в даній роботі.

4.1.2 Вибір метрики для оцінки моделі машинного навчання

Вибір правильної метрики відіграє ключову роль в задачі оцінки моделей машинного навчання. Для оцінки моделей ML в різних програмах пропонуються різні метрики. На деяких проектах стається так, що експерти дивляться лише на одну певну метрику, але через це є реальна загроза не представити всю картину проблеми в цілому. Варто також пам’ятати, що метрика сама по собі відрізняється від функції втрат. Функції втрати – це функції, які показують міру продуктивності моделі і використовуються для навчання моделі машинного навчання (використовуючи певний алгоритм оптимізації) і зазвичай відрізняються від параметрів моделі. З іншого боку, метрика використовується для моніторингу та вимірювання продуктивності моделі (під час процесу тренування та тестування), і їх потрібно розглядати окремо. Однак якщо для деяких завдань функція продуктивності є диференційованою, її можна використовувати також в якості функції втрат (можливо, додавши до нього деякі регуляризації), так і в якості метрики, наприклад, середньоквадратична помилка.

Задача класифікації стала однією з найбільш широко розповсюджених проблем у машинному навчанні через те, що існує купа різних промислових застосувань таким алгоритмам: від розпізнавання облич, пропонування відео на Youtube, модерація

повідомлень, медична діагностика, до класифікації тексту, виявлення прикладів ненависті на Twitter тощо. Тому без правильно обраних метрик неможливо уявити, щоб такі гіганти досягли великих успіхів.

Матриця помилок являє собою табличну візуалізацію прогнозів моделі, порівняно з очікуваними істинними відповідями. Кожен рядок матриці помилок представляє собою екземпляри в передбачуваному класі, а кожен стовпець представляє екземпляри фактичного класу. Наприклад, якщо стоїть задача двійкової класифікації для пошуку зображень з котами та таких, на яких немає котів, то матриця матиме вигляд матриці розміром 2 на 2 клітинки, де по горизонталі будуть записуватися кількості очікуваних значень, а по вертикалі – значення, що були надані на вихід моделлю машинного навчання. При чому і по вертикалі, і по горизонталі існує лише дві можливі відповіді – «кіт» або «не кіт». Таким чином, діагональні елементи цієї матриці позначають кількість правильних передбачень для різних класів, тоді як елементи, що не лежать на діагоналі, позначають відповіді, які класифікуються неправильно.

Наступна метрика – точність класифікації – це мабуть найпростіша метрика, яку можна уявити, і рахується вона як кількість правильних прогнозів, розділених на загальну кількість прогнозів, помножених на 100%. Хоч це і проста та інтуїтивно зрозуміла метрика, використовувати її можна лише на збалансованих даних, тобто коли кількість значень різних класів однакова або майже однакова.

Метрика правильності використовується в тому випадку, коли звичайна точність класифікації не є хорошим показником продуктивності моделі. Один із таких сценаріїв – коли розподіл значень класів у наборі даних незбалансований (один клас трапляється частіше за інші). У цьому випадку, навіть якщо спрогнозувати всі зразки як найчастіший клас (тобто модель перенавчилася на одному класі), можна все одно отримати високу точності, яка втім зовсім не має сенсу. Тому потрібно також переглянути конкретні показники ефективності для класу. Правильність – один з таких показників, який визначається за формулою 4.1:

$$Precision = \frac{TP}{TP+FP}, \quad (4.1)$$

де Precision – це значення правильності, TP – кількість об’єктів, які модель правильно віднесла до позитивного класу, а FP – кількість об’єктів, які модель віднесла до негативного класу, але неправильно. Ще одна метрика, яку часто використовують для оцінки моделей це відклик. Відклик визначається за формулою 4.2, як частка вибірки класів, які правильно прогнозуються моделлю:

$$Recall = \frac{TP}{TP+FN}, \quad (4.2)$$

Де Recall – це значення величини відклику, а TP, як вже зазначалось, це кількість об’єктів, які модель правильно віднесла до позитивного класу, а FN – це кількість об’єктів, які були віднесені моделлю до негативного класу, але таке твердження неправильне.

Залежно від застосування, можливо, існує потреба у науковців надати більш високий пріоритет правильності або відклику. Але є багато прикладних застосувань, в яких важливе значення мають як правильності, так і відклику. Одна з найпопулярніших метрик, яка поєднує в собі правильність та відклик, називається F1-оцінкою, яка розраховується як середнє гармонічне між значеннями правильності та відклику, і визначається за формулою 4.3:

$$F1 = 2 * \frac{Precision*Recall}{Precision+Recall}, \quad (4.3)$$

Де F1 – це значення оцінки, precision – це значення правильності, а recall – це значення відклику. Варто зазначити, що завжди існує компроміс між точністю і відкликом моделі, якщо точність моделі стає занадто високою, можна побачити падіння значення виклику і навпаки.

Крива робочої характеристики приймача ROC – це графік, який показує продуктивність двійкового класифікатора як функції його порогу відсічення. Дана метрика по суті протиставляє значення справжньої позитивної частки (TPR) проти хибної позитивної частки (FPR) для різних порогових значень. Оскільки багато моделей машинного навчання для задач класифікації є ймовірнісними, вони передбачають деяку ймовірність того, що зразок належить до певного класу. Потім вони порівнюють цю вихідну ймовірність з деяким порогом відсічення, і якщо таке вихідне значення перевищує такий поріг, то клас визначається як позитивний.

Таким чином, змінюючи порогові значення, можна отримати різні значення для різних міток. І як можна собі уявити, кожен із цих сценаріїв призведе до різних значень правильності та відклику. Крива ROC, зображена на рисунку 4.1, по суті виявляє співвідношення TPR і FPR для різних порогових значень. На графіку TPR визначається по осі У, а FPR по осі Х.

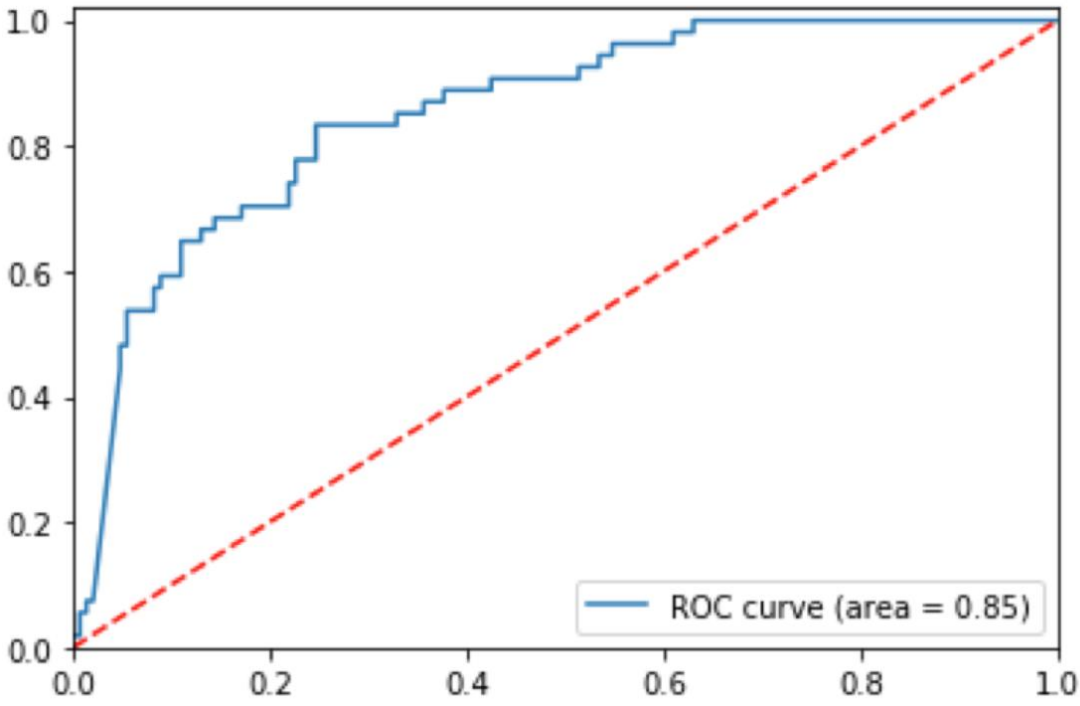

Рисунок 4.1 – Приклад графіку кривої ROC

Як видно з графіку, чим нижчий поріг граничного відхилення для позитивного класу, тим більше вибірок прогнозується як позитивний клас, тобто вищий істинний

позитивний показник, відомий ще як відклик, а також вищий хибний позитивний показник (відповідає правій частині даної кривої). Таким чином, існує компроміс між тим, наскільки високим може бути відклик, і наскільки потрібно обмежити помилку (FPR). Крива ROC – популярний інструмент, який допомагає вивчити загальну продуктивність моделі та вибрати хороший поріг відсічення для моделі.

Метрика площа під кривою (AUC) – це сукупний показник продуктивності двійкового класифікатора для всіх можливих порогових значень. AUC обчислюється, як площа під кривою ROC, і тому її значення знаходяться в межах між 0 і 1. В загальному випадку, чим вищі значення метрик AUC моделі машинного навчання, тим вона краща. Можна навіть налаштувати порогове значення для моделі таким чином, щоб він відповідав мінімальним вимогам щодо цих показників.

Тому, щоб вирішити, як краще за все оцінювати ефективність класифікаційної моделі, спершу найкраще зрозуміти вимогу бізнесу та вирішити, для якої метрики краще за все оптимізувати модель. З практичної точки зору, класифікаційна модель, яка видає ймовірності, є кращою перед виведенням одного результату, оскільки це забезпечує гнучкість настройки порогу таким чином, щоб він відповідав вашим мінімальним вимогам до відклику або правильності.

Оскільки для розглянутої задачі було вирішено використовувати модель рекурентної нейронної мережі, яка вирішує задачу класифікації, то було вирішено використовувати наступні метрики для оцінки якості моделі: оцінка f1 та AUC ROC. В нашому випадку варто зазначити, що дані є незбалансованими, а тому не можна було використовувати звичайні метрики, такі як точність. Такі метрики було обрано з багатьох причин, і незбалансованість даних – одна з них. Проблема також полягає в тому, що об’єкти, які було класифіковано правильно моделлю, мають таке саме значення, що й об’єкти, які було класифіковано неправильно. В такому випадку метрики f1 та AUC ROC чудово справляються з завданням правильної оцінки моделі.

4.2 Пошук оптимальних параметрів алгоритмів машинного навчання

Процес навчання моделі машинного навчання може складатися з декількох етапів, які обов’язково повинні виконуватися в строгій послідовності, адже виконання наступного етапу напряму залежить від виконання попереднього. Загалом існують різні підходи до впровадження моделей машинного навчання у виробництво, причому переваги можуть змінюватися залежно від конкретного випадку їх використання. Наприклад, розділяють два окремих процеси – навчання моделей та використання моделей для створення передбачень. Кожен з таких процесів також можна реалізувати по-різному.

4.2.1 Вибір архітектури для підтримки моделі

Для такого підходу навчання моделей, коли модель навчається лише один раз, модель може бути навчена та точно налаштована експертами з машинного навчання або бібліотекою AutoML. Також є декілька способів проведення навчання моделей. Один з таких способів це серійне навчання. Якщо не стоїть серйозна потреба для використання моделі на виробництві, то такого підходу буде більш, ніж достатньо. Підхід до навчання моделей на серіях даних надає можливість мати свіжу версію моделей одразу ж після останнього процесу тренування.

Такий підхід можна дуже спростити, якщо використовувати фреймворки, які працюють за принципом AutoML, які дозволяють автоматизувати такі процеси, як обробка даних, створення нових колонок даних, вибірка колонок даних, вибір моделі та оптимізація параметрів. Існують різні технології, які підтримують таке постійне навчання на основі серії даних; вони, наприклад, можуть бути налаштовані за допомогою поєднання різних фреймворків для управління робочими процесами та бібліотекою AutoML. Різні постачальники хмарних сервісів пропонують свої рішення для AutoML, якими можна замінити користувацький процес обробки даних. Наприклад, Azure інтегрує прогнозування машинного навчання та модельне навчання зі своїми внутрішніми даними для отримання кращого результату.

На противагу такому підходу серійного навчання моделі стоїть навчання в режимі реального часу. Такий підхід став можливим після створення відповідних моделей машинного навчання, що підтримують такий метод навчання. До таких алгоритмів можна віднести K-середніх, лінійну та логістичну регресію (через стохастичний градієнтний спуск), а також наївний баєсовий класифікатор.

Під час розгортання такого типу моделей на виробництві необхідна серйозна оперативна підтримка та моніторинг, оскільки модель може бути чутливою до нових даних і шуму, а продуктивність моделі повинна контролюватися постійно. У режимі офлайн-навчання можна фільтрувати дані з високим рівнем впливу та виправляти такий тип вхідних даних, видаляючи їх або коригуючи. Проте це набагато складніше робити, коли відбувається постійне оновлення моделей машинного навчання на основі потоку нових даних.

Інша проблема, яка виникає при навчанні онлайн-моделі, полягає в тому, що такі моделі складно переналаштовувати для використання з іншим видом даних. Це означає, що, якщо у наборі даних відбудуться структурні зміни, модель потрібно буде вкінці-кінців переробити та навчати заново, що матиме великий вплив на управління життєвим циклом моделі в подальшому.

Намагаючись зрозуміти чи потрібно використовувати на виробництві пакетне прогнозування чи прогнозування в реальному часі, важливо, щоб спеціаліст усвідомлював, навіщо саме робити прогнозування в реальному часі. Потрібно замислитися чи, це не призведе до погіршення задоволеності клієнтів, чи не погіршиться оцінка клієнта продукту, коли той телефонує у контактний центр. Ці переваги необхідно зважити та проаналізувати разом зі значенням складності та вартості, які виникають при створенні моделі прогнозування в реальному часі.

Створення прогнозу за допомогою моделі, що працює в режимі реального часу вимагає пошуку способу впоратися з піковими навантаженнями. Залежно від підходу, що було обрано вкінці-кінців та способу використання прогнозування, для підтримки підходу прогнозування в реальному часі також може знадобитися наявність сервера з додатковою обчислювальною потужністю для того, щоб забезпечити прогнозування

в межах певної рівень обслуговування. В цьому полягає різниця між таким і пакетним підходом, коли обчислення прогнозів можуть бути розподілені протягом дня на основі наявної потужності.

Коли система працює в реальному часі, значно зростає потреба в ресурсах та часу відгуку системи. Експертам потрібно бути в курсі працездатності системи, вони бажають бути проінформованими кожного разу, коли з якоїсь причини система починає відмовляти. Для підходу, коли передбачення генеруються пакетно, такі потреби значно знижуються, а будь-які критичні ситуації, що відбуваються під час цього процесу не впливають на задоволення користувача від використання системи.

Також варто зазначити, що потреба в передбаченнях в реальному часі впливає на витрати підприємства, адже зі зростанням використовуваних ресурсів протягом дня, за які відповідно потрібно платити, збільшує загальні витрати компанії на використання моделей. Зважаючи на обраний підхід та вимоги, поставлені перед компанією, додаткові витрати можуть також з’являтися задля задоволення рівня надання послуг IT-інфраструктурами.

А задля того, щоб порахувати метрики, наскільки якісно відпрацьовує модель, у реальному часі знадобилося б більше ресурсів. Оскільки така потреба виникає в реальному часі, дані, які підтверджують або спростовують передбачення також приходять у реальному часі. Також для застосування такого підходу потрібно мати комплексний механізм обробки логів та подій.

Задля інтеграції алгоритмів прогнозування з використанням пакетного прогнозу потрібно розуміти, що такі прогнозування покладаються на два різних набори даних, один – власне модель прогнозування, а другий – особливі ознаки даних, які ми будемо надсилати до моделі. У більшості типів архітектури пакетного прогнозування для отримання заздалегідь обчислених функцій з конкретного сховища даних або виконання певного типу перетворень у декількох наборах даних, щоб забезпечити передачу даних до моделі прогнозування, використовується підхід ETL. Потім модель прогнозування перебирає кожний рядок в цих наборах даних, рахує помилку та надає оцінку згідно з метриками.

Після того, як всі прогнози були обчислені, ми можемо порахувати оцінку моделі та надіслати її різним системам, які агрегують інформацію. Це може бути зроблено по-різному, залежно від ситуації, для якої ми хочемо дати оцінку. Наприклад, якби ми хотіли скористатись розрахованою відповіддю моделі на стороні користувача, ми, швидше за все, перенесемо дані в кеш або база даних NoSQL, наприклад Redis, що дозволило б надавати відповіді користувачу протягом мілісекунд, тоді як для певних випадків використання, таких як надсилання електронного листа, ми можемо просто покластися на інструменти для CSV-експорту SFTP або завантаження даних у більш традиційні RDBMS.

Щоб мати можливість використовувати модель у виробництві для додатків у режимі реального часу, потрібні 3 базові компоненти. Профіль клієнта або користувача, набір тригерів та власне моделі прогнозування.

Профіль клієнта містить усі пов'язані з ним атрибути, а також різні атрибути (наприклад, лічильники), необхідні для того, щоб зробити заданий прогноз. Це потрібно для прогнозування клієнтської інформації, щоб зменшити затримку завантаження інформації з декількох місць, а також спростити інтеграцію моделей машинного навчання у виробництві. У більшості випадків схожий тип сховища даних знадобиться для ефективного отримання даних, необхідних для навчання моделі прогнозування.

Тригери – це певні події, що ініціюють процеси, вони можуть бути, наприклад, викликом, зателефонувати в центр обслуговування клієнтів, перевірити інформацію в історії замовлення тощо.

Моделі: моделі повинні бути попередньо підготовлені і, як правило, експортуватися в один із 3 згаданих раніше форматів (архівація, ONNX або PMML), щоб їх було легко перенести у виробництво.

Задля використання моделей машинного навчання на виробництві було створено різні підходи:

- наприклад, постачальники програмних продуктів для управління базами даних, докладають великих зусиль для того, щоб інтегрувати аналітичні інструменти прямо в їх платформу;
- використовуючи підхід «підписник-видавець», модель машинного навчання просто отримує на вхід потік даних та виконує певні операції, наприклад, скачування інформації про профіль користувача;
- веб-сервіси дозволять створити прикладний програмний інтерфейс для моделі машинного навчання та розгортання моделі як сервіс;
- вбудований підхід дозволяє розгортати моделі напряму в користувацьких застосунках, що дає можливість моделям запускатися на устаткування користувача, не використовуючи ресурси розробника.

Кожен з таких підходів має свої переваги та недоліки. Тому задля створення та підтримки моделі, що описана в даній роботі, було вирішено використовувати гібридний підхід, який комбінує в собі сильні сторони підходу з використанням центральної бази даних для зберігання даних та моделей, підхід «підписник-видавець» використовується для створення передбачень в реальному часі, але при цьому модель тренується та налаштовується паралельно і окремо від основної модель, що використовується для передбачень.

4.2.2 Пошук гіперпараметрів моделі машинного навчання

Моделі машинного навчання складаються з двох різних типів параметрів: гіперпараметри – це всі параметри, які користувач може довільно задати перед початком навчання (наприклад, кількість оцінювачів у алгоритмі випадкового лісу). Параметри моделі – вивчаються замість цього під час навчання моделі (наприклад, ваги в нейронних мережах, лінійній регресії). Параметри моделі визначають, як використовувати вхідні дані для отримання бажаного результату та визначаються під час навчання. Натомість гіперпараметри визначають, яку структуру матиме алгоритм в першу чергу.

Налаштування моделей машинного навчання відноситься до проблем оптимізації [18]. Спочатку заданий набір гіперпараметрів і користувач прагне знайти правильну комбінацію їх значень, яка може допомогти знайти або мінімум (наприклад, значення функції втрат), або максимум (наприклад, точність) функції. Це питання може бути особливо важливим при порівнянні ефективності різних моделей машинного навчання на наборах даних. Насправді, було б несправедливо, наприклад, порівнювати модель SVM з найкращими гіперпараметрами з моделлю випадкового лісу, яка не була оптимізована.

Для початку процесу оптимізації потрібно визначити базове значення якості моделі. Отримавши певне початкове значення якості моделі, можна використовувати різні підходи, щоб збільшувати це значення або зменшувати. Найпростішим є підхід ручного пошуку гіперпараметрів. Під час використання ручного пошуку ми вибираємо деякі гіперпараметри моделі, виходячи з наших суджень або досвіду. Потім ми тренуємо модель, оцінюємо її точність і запускаємо процес заново. Ця процес повторюється доти, поки не буде отримано задовільної точності.

Випадковий пошук гіперпараметрів це ще один підхід, при якому спочатку створюється сітка гіперпараметрів – це комбінація деяких значень гіперпараметрів у багатовимірному просторі – і тренуємо або тестуємо нашу модель лише на деякій випадковій комбінації цих гіперпараметрів. Вирішуючи задачу машинного навчання, як вже було зазначено раніше, зазвичай набір даних ділиться на навчальну та тестову вибірки. Це робиться для того, щоб можна було протестувати модель після її навчання, таким чином є змога перевірити ефективність моделі машинного навчання під час роботи з небаченими даними. Під час використання підходу, що дістав назву крос-валідації навчальний набір даних ділиться на ще інші розділи, щоб переконатися, що модель не перенавчається на одному наборі даних.

Один з найпоширеніших методів крос-валідації – це валідація K-Fold. Використовуючи K-Fold валідацію навчальний набір даних ділиться на N розділів, а потім модель ітеративно тренується на окремих розділах даних по кількості N-1 і тестується на іншій частині даних (під час кожної ітерації змінюються набори даних).

Після тренування моделі N разів можна потім усереднити оцінку результатів роботи моделей, отримані за кожну ітерацію, щоб отримати загальні результати тренування моделей машинного навчання.

Пошук за допомогою сітки гіперпараметрів це ще один спосіб оптимізації моделей. У такому виді пошуку створюється сітка з гіперпараметрів, далі модель тренується, а потім тестується на кожній з можливих комбінацій таких параметрів. Для того, щоб обрати параметри, які використовуватимуться в пошуку за допомогою сітки, можна візуалізувати дані про те, які параметри найкраще всього працювали з випадковим пошуком і сформувати на їх основі сітку, щоб побачити, чи можливо знайти ще кращу комбінацію.

Пошук гіперпараметрів за допомогою сітки повільніший, порівняно з випадковим пошуком, але він може бути в цілому більш ефективним, оскільки може пройти через весь простір пошуку. Натомість випадковий пошук може бути більш швидким, але може пропустити деякі важливі значення параметрів в просторі пошуку.

Автоматизоване налаштування гіперпарамтрів є більш просунутою технікою оптимізації моделі. При використанні автоматизованої настройки гіперпараметрів моделі, які використовуються моделями, визначаються за допомогою таких методів, як: Байєсова оптимізація, градієнтний спуск та генетичні алгоритми.

Байєсова оптимізація може бути використана за допомогою мови Python [16], де вона реалізована в бібліотеці Hyperopt. Байєсова оптимізація використовує ймовірнісну інформацію для того, щоб знайти мінімум функції. Кінцева мета цього алгоритму – це знайти таке вхідне значення функції, яке може дати найменше можливе вихідне значення. Байесівська оптимізація виявилася ефективнішою, ніж випадковий, сітковий або ручний пошук. Таким чином, використовуючи таку оптимізацію можна отримати кращі показники на етапі тестування та скорочення часу оптимізації.

Аби оптимізувати значення функції за допомогою бібліотеки Hyperopt, Байєсова оптимізація може бути реалізована, надаючи 3 основні параметри функції

fmin(). Цільова функція визначає функцію втрат для мінімізації. Доменний простір визначає діапазон вхідних значень для тестування (в Байєсовій оптимізації цей простір створює розподіл ймовірностей для кожного з використаних гіперпараметрів). Алгоритм оптимізації визначає алгоритм пошуку, який слід використовувати для вибору найкращих вхідних значень, які слід використовувати для кожної нової ітерації. Байєсова оптимізація може зменшити кількість ітерацій пошуку шляхом вибору вхідних значень, враховуючи результати попередніх проходів. Таким чином ми можемо з самого початку сконцентрувати пошук на значеннях, ближчих до бажаного результату.

Генетичні алгоритми намагаються застосувати природні механізми відбору в контексті машинного навчання. Вони натхненні дарвінським процесом природного відбору, і тому їх також інколи називають еволюційними алгоритмами. Наприклад, якщо створено сукупність з N моделей машинного навчання з деякими заздалегідь визначеними гіперпараметрами, то можна обчислити точність кожної моделі та зберегти лише половину моделей (тих, які найкраще працюють та дають найкращий результат). Далі можна згенерувати кілька нащадків таких моделей, які будуть мати подібні гіперпараметри до найкращих моделей, щоб знову отримати сукупність N моделей. На цьому етапі можна знову обчислити точність кожної моделі і повторити цикл на певну кількість поколінь. Таким чином, лише найкращі моделі «виживуть» в кінці процесу.

Оскільки в роботі було вирішено використовувати модель машинного навчання рекурентна нейронна мережа, то процес оптимізації моделі проходив у декілька етапів. По-перше, варто зазначити, що оскільки така модель є дуже складною, існує багато гіперпараметрів, які потрібно оптимізовувати. До таких гіперпараметрів входять:

- кількість прихованих шарів нейронної мережі;
- кількість нейронів в кожному шарі;
- кількість ітерацій алгоритму оптимізації;
- параметри, які використовуються для знаходження градієнтів;

- алгоритми оптимізації моделі;
- використання різних функцій активації.

Завдяки набору різних алгоритмів, було підібрано таку комбінацію параметрів, яка дала змогу отримати найкраще значення метрики якості моделі машинного навчання серед усіх варіантів.

4.3 Тестування ефективності розробленого рішення

Оскільки модель машинного навчання, яка була натренована на реальних даних компанії, була недоступна до використання в ході експерименту через підписаний раніше договір про нерозголошення корпоративних таємниць, дані для тренування моделі було згенеровано локально за допомогою вже існуючих відкритих наборів даних. Дані про використання ресурсів компонентами критичної IT-інфраструктури було згенеровано в тестовому середовищі з використанням високорівневої мови програмування Python. Дані було збережено в єдине сховище, але для простоти аналізу та тренування експериментальної моделі їх було розбито на проміжки довжиною в п’ять секунд.

Варто зазначити, що згенеровані дані є «чистими», тобто вони не мають не значень, які критично відрізняються від середньостатистичних значень по усьому набору даних. Також згенеровані дані, як показано на рисунку 4.2, не мали відсутніх значень, але в корпоративному середовищі ситуація зовсім інша – на виробництві дані, що отримуються від компонентів системи можуть бути відсутніми протягом деякого проміжку часу, що ускладнює процес моніторингу. Також в згенерованих даних точки даних зустрічаються з однаковою частотою протягом усього часового проміжку наявних даних. На виробництві дані бувають розподілені нерівномірно, тому доводиться на практиці використовувати методи екстраполяції, що дозволяють заповнити прогалини в даних.

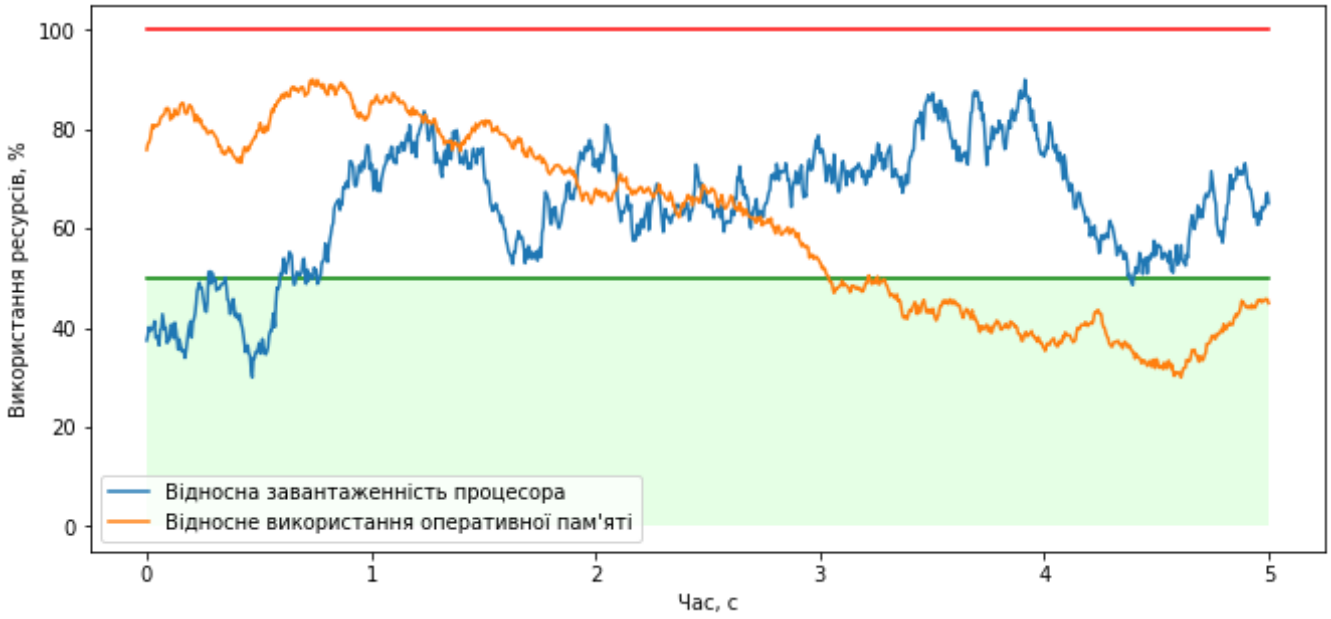

Рисунок 4.2 – Графік використання ресурсів компонентами критичної IT-інфраструктури

Згенеровані дані відображають відносне використання двох найбільш популярних для моніторингу типів ресурсів в критичній IT-інфраструктурі – оперативної пам’яті та процесорного часу. Задля тренування моделі всі наявні дані було поділено на часові проміжки довжиною в п’ять секунд. Кожен з таких часових проміжків міг містити інформацію про помилку або не містити її. Кількість таких часових відрізків з наявною помилкою та кількість всіх інших співвідносилися як один до трьох.

Деякі часові проміжки мали інформацію про збільшене використання ресурсів. Такі проміжки зображені на рисунку 4.3. Червоним позначена зона збільшеного використання ресурсів, а зеленим – зона, в якій компонент системи використовує менше ресурсів, ніж є наявними в системі. Так на рисунку видно, що в першому випадку хоча спочатку відносна завантаженість процесора була невисокою, менше, ніж 50% процесорного часу, як і оперативної пам’яті, було використано, проте використання ресурсів збільшилося суттєво протягом лише двох секунд. Це могло виникнути як наслідок запуску підпроцесів системи, наприклад індексації бази даних або збільшення потоку клієнтів на ресурс.

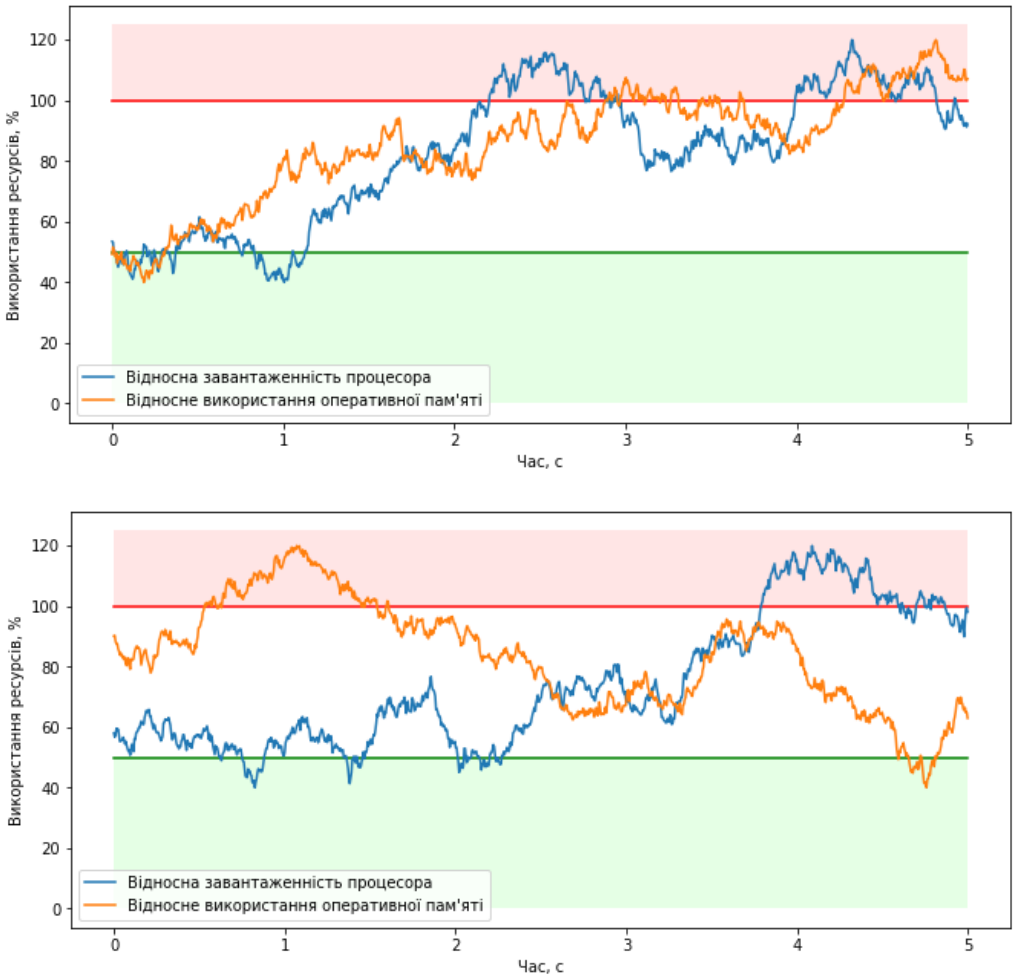

Рисунок 4.3 – Проміжки збільшеного використання ресурсів.

Така ситуація, коли багато ресурсів використовується системою, може призвести до збільшення часу відгуку сервісів компанії, що може призвести до формування негативної думки з боку клієнта. Щоб такого не допустити, натренована модель машинного навчання повинна вміти розрізняти потенційно небезпечні ситуації заздалегідь та намагатися їх виправити.

Оскільки доволі складно знайти таку вигляд даних, який би давав всебічну оцінку стану системи, було вирішено спробувати два підходи до вивчення моделлю машинного навчання співвідношень вхідних та вихідних даних. За першим принципом, вхідним даним ставиться у відповідність вихідні дані, які включають інформацію про факт наявності помилки. Якщо факт наявності помилки був позитивним, то окрім цього до вихідного вектора включається інформація про те, яку

дію потрібно застосувати для виправлення помилки, та скільки потрібно додаткових ресурсів виділяти.

Ще один спосіб полягає в тому, що вхідним даним про стан системи у відповідність ставились дані про відсоток імовірності наявності помилки, що вимірюються значеннями на проміжку від 0 до 1. Також до цього додається інформація про те, яких ресурсів не вистачає або навпаки є надлишок і чи потрібно вимкнути даний вузол, наприклад, якщо ресурсів занадто багато. Такий підхід виявся більш корисним, адже надає змогу одночасно аналізувати наявність декількох ресурсів.

Окремо варто виділити процес пошуку гіперпараметрів, який передував основному процесу навчання моделі. Пошук гіперпараметрів здійснювався комбінованим методом, як було написано вище, по таким параметрам, як кількість шарів нейронної мережі, кількість нейронів в кожному шарі, параметри регуляризації та оптимізації тощо.

Модель машинного навчання рекурентну нейронну мережу було натреновано на персональному комп’ютері з вбудованою відеокартою Nvidia Geforce GT740M, що є бюджетною версією повноцінної відеокарти для персональних комп’ютерів. Дана відеокарта містить 234 CUDA-ядра, які дозволяють вести паралельні розрахунки і проводити операції над матрицями та векторами. Наявність відеокарти навіть невисокого рівня дає прискорення процесу тренування, що може вимірюватися десятками разів. Через це середня швидкість тренування однієї епохи моделі рекурентної мережі, зайняло в середньому 20 секунд, а загалом процес тренування та валідації моделі на всіх наявних даних – 1000 проміжків по п’ять секунд, зайняло близько двох годин. Логи тренування приведено на рисунку 4.4.

```
Started training GRUCell for 500 epochs
Initial loss 6.597514453040242
Iteration 50/500 progress: [================>] 100% | loss: 1.7775790336033697
Iteration 100/500 progress: [================>] 100% | loss: 1.4071286543997494
Iteration 150/500 progress: [================>] 100% | loss: 1.289517392520851
Iteration 200/500 progress: [================>] 100% | loss: 1.2152960701290918
Iteration 250/500 progress: [================>] 100% | loss: 1.1624809088290657
Iteration 300/500 progress: [================>] 100% | loss: 1.1166687317720194
Iteration 350/500 progress: [================>] 100% | loss: 1.1270043478966205
Iteration 400/500 progress: [================>] 100% | loss: 1.1403739800753314
Iteration 450/500 progress: [================>] 100% | loss: 1.091131292156655
Iteration 500/500 progress: [================>] 100% | loss: 1.0926277130406798
Finished training for 500 epochs, final loss: 1.0926277130406798
Validated with f1-score metric value: 0.8609813274
```

Рисунок 4.4 – Логи тренування моделі машинного навчання

В ході тренування було отримано значення функції втрат для кожної епохи тренування. З логів видно, що значення функції втрат зменшувалося протягом перших 450 епох, проте під кінець останньої епохи тренування значення незначно збільшилося. Це можна пояснити тим, що нейронна мережа почала перенавчатися, що могло негативно сказатися на результатах тестування в подальшому. Найкращий вихід в такій ситуації – це зупинити навчання моделі в переламний момент, коли значення функції втрат починає зростати, як можна бачити на рисунку 4.5.

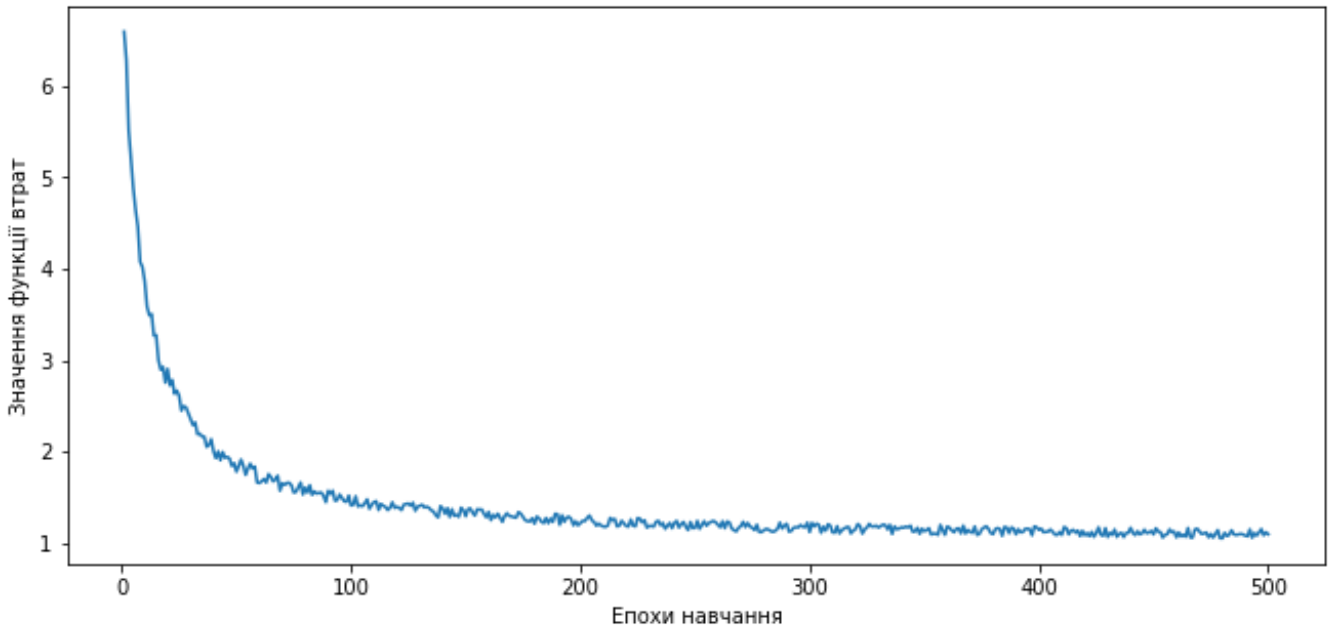

Рисунок 4.5 – Графік функції втрат навчання моделі нейронної мережі

Проте в даному випадку ріст функції витрат є невеликим, і це можна пов’язати зі стохастичною природою алгоритмів машинного навчання. Фінальне значення функції втрат відносно невисоке – 1.0927. Значення метрики f1 після перевірки на тестувальному наборі даних становило 0.86, що є непоганим результатом, враховуючи природу даних та обмежену їх кількість. Навіть з таким значенням даної метрики, модель все одно справлятиметься з задачею передбачення стану системи.

Значення метрики ROC зображено на рисунку 4.6. З графіку видно, що модель відпрацьовує набагато краще, ніж базова оцінка – тобто просто довільні передбачення. Площа під кривою доволі велика – 0.87, що також говорить про високу якість моделі.

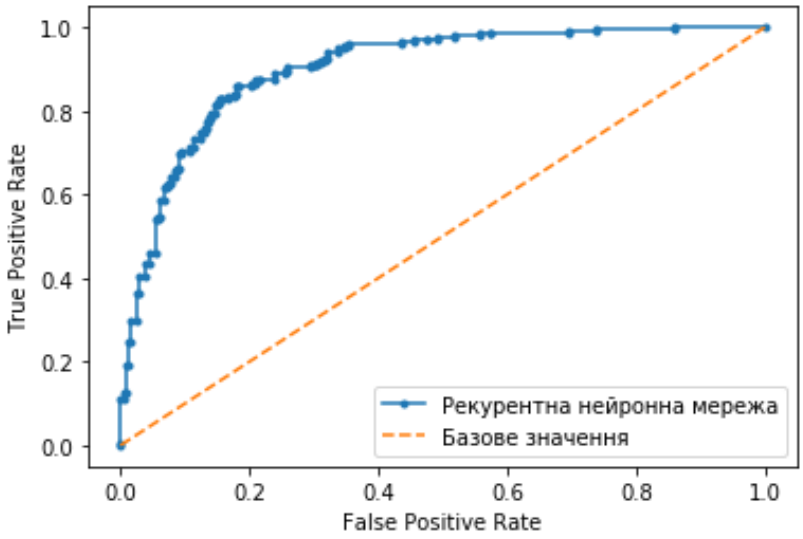

Рисунок 4.6 – Графік ROC для моделі машинного навчання в порівнянні з базовим значенням

Валідація результату відбувається на окремому наборі даних. Для цього для відрізку даних створюється передбачення щодо управління ресурсами. Приклад отриманого результату приведений на рисунку 4.7. На рисунку видно, що в чотирьох точках на графіку рекурентна нейронна мережа визначила місця для потенційного масштабування.

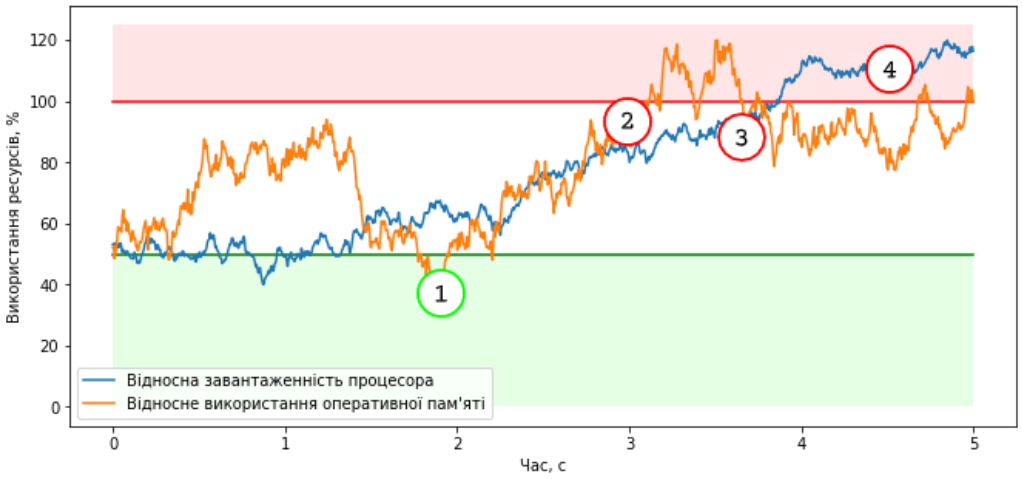

Рисунок 4.7 – Приклад передбачення моделі машинного навчання

Точка 1 була визначена системою, як точка, в якій потрібно змінити налаштування оперативної пам’яті, а саме зменшити кількість виділеної пам’яті для даного компонента критичної IT-інфраструктури на 15 відсотків. Проте вже в точці 2 системи побачила проблему у зв’язку зі збільшенням використання оперативної пам’яті. В даній точці модель рекомендувала збільшити кількість виділеної оперативної пам’яті на 20%, що, як видно з графіку, було дуже доречно через деяку кількість часу.

З іншого боку, зросло також використання процесорного часу, що потенційно може загрожувати збільшенням часу відгуку системи. В точці 3 моделлю було передбачене збільшення використання ресурсу процесора та рекомендовано збільшити кількість цього ресурсу на 20%. В точці 4 модель знову оцінила стан використання ресурсу процесора, як загрозливий для відгуку системи і рекомендувала збільшити ще на 10% кількість цього ресурсу.

Після отримання прогнозу відбувається розшифровка результатів. Якщо кількість ресурсів, яка потрібна для масштабування системи більше за певне порогове значення, визначене адміністратором, то подається сигнал про перерозподіл ресурсів між вузлами з подальшим створенням нового вузла системи.

4.4 Висновки до розділу

В даному розділі було визначено кроки, які необхідні для підготовки до навчання моделі штучної нейронної мережі: вибір метрики для перевірки якості моделі, створення набору даних, на яких буде навчатися алгоритм та налаштування гіперпараметрів. Було обрано відкритий стандарт для збереження моделей машинного навчання ONNX, що дозволяє конвертувати моделі у представлення, які можна використовувати на різних платформах та зберігати моделі у базу даних.

Також було визначено найкращий спосіб поділу даних на тренувальну та тестувальну вибірки. Тренувальна вибірка використовувалася під час тренування моделі машинного навчання та складала більшу частину загального набору даних. Тестувальна вибірка даних використовувалася на етапі отримання оцінки моделі машинного навчання. В ході навчання було також використано підхід перехресної валідації даних під назвою K-Folds. Використання поділу даних, хоч і сповільнило процес навчання моделі, проте надало моделі можливість тренувати на нових комбінація даних щоразу, що значно зменшило вірогідність того, що модель перенавчиться.

Для оцінки якості моделі було вирішено використовувати метрики для оцінки якості класифікації – оцінка f1 та AUC ROC. Такі метрики найкраще відображають відношення правильності та точності, а також їх можна використовувати у випадку незбалансованості даних, що є актуальним для даної роботи.

Було проведено пошук гіперпараметрів моделі рекурентної нейронної мережі, таких як кількість прихованих шарів нейронної мережі, кількість нейронів в кожному шарі, кількість ітерацій алгоритму оптимізації, параметри, які використовуються для знаходження градієнтів тощо. Після знаходження оптимальної комбінації гіперпараметрів моделі машинного навчання, рекурентну нейронну мережу було натреновано на даних для тренування.

Модель було натреновано протягом 500 епох на 1000 проміжках згенерованих даних. Значення функції втрат зменшилося з початкового значення з 6.6 до 1.09, оцінка f1 склала 0.86, а значення AUC ROC – 0.87, що є хорошим результатом.

5 РОЗРОБКА СТАРТАП ПРОЕКТУ

Стартап – це такий вид бізнесу, основою якого є інноваційність, яка є підґрунтям для вирішення проблеми шляхом створення нового продукту чи послуги в умовах крайньої невизначеності. Багато підприємців та відомих ділових магнатів визначають стартап як культуру та менталітет будувати бізнес на основі інноваційнійої ідеї для вирішення критичних моментів, які ще не були вирішені.

Єдина ознака, що відрізняє стартапи від інших видів підприємств, - це співвідношення між продуктом, які вони виробляють, та попитом на нього. У стартапів продукти, націлені на ринок, на якому досі не було схожих продуктів, тому існує великий попит. Зазвичай для створення стартапу обирається стратегія розробки такого продукту, який потребує ринок, з наступним виходом на ринок та його захопленням. Така стратегія забезпечує швидке зростання стартапу.

В даному розділі проводиться розробка стартап проекту системи управління якістю надання послуг критичними IT-інфраструктурами, проводиться маркетинговий аналіз ринку, аналіз слабких та сильних сторін проекту, аналіз конкурентів тощо.

5.1 Опис ідеї проекту

Ідеєю стартап проекту, розробленого в даній дисертації є створення системи управління якістю надання послуг критичними IT-інфраструктурами. Така система повинна давати можливість підприємствам з наявною критичною IT-інфраструктурою, збирати дані про роботу системи, проводити моніторинг та за потреби передбачати можливі несправності в майбутньому. Продукт повинен бути зручним в користуванні для адміністраторів системи та надавати якомога більше інформації про стан системи в графічному вигляді. При виявленні можливих несправностей, система повинна терміново попереджувати адміністратора системи через доступні канали зв’язку та виводити в користувацький інтерфейс такі попередження. Опис ідеї стартап-проекту приведено в таблиці 5.1.

Таблиця 5.1 – Опис ідеї стартап-проекту

Зміст ідеї	*Напрямки застосування*	*Вигоди для користувача*
	1. Оптимізація використання ресурсів IT-інфраструктури	
	2. Предиктивна аналітика стану критичної IT-інфраструктури	

В ході аналізу ринку було виявлено два основних конкуренти даного продукту, які мають схожі характеристики та схожі цілі. До переваг програмного забезпечення SignalFx можна віднести моніторинг у реальному часі та інтелектуальний аналіз середовища Kubernetes та робочих навантажень серверів IT-інфраструктури, надаючи власникам сервісів та операторам інфраструктури інформацію, яка їм потрібна для визначення проблеми та знаходження відповідного рішення заздалегідь, перш ніж ці проблеми вплинуть на клієнтів.

До переваг продукту Datadog можна віднести наявність відкритого коду, підтримки з боку розробників та громади, що робить даний продукт хорошим інструментом для управління якістю надання послуг критичною IT-інфраструктурою. А наявність плагінів, які надають можливість системі керувати ресурсами в режимі реального часу лише збільшує корисність такого додатку.

Проте, суттєвим недоліком обох додатків є те, що вони працюють лише з хмарною інфраструктурою та можуть бути використані виключно для середовища Kubernetes. В таблиці 5.2 було визначено сильні, слабкі та нейтральні характеристики ідеї проекту.

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

п/п	*Техніко-економічні характеристики ідеї*	*(потенційні) товари/концепції конкурентів*			*W (слабка сторона)*	*N (нейтральна сторона)*	*S (сильна сторона)*
		Система управління якістю надання послуг	*SignalFx*	*Datadog*			
1.	Вартість	Середня	Висока	Середня			+
2.	Можливість обробки різних видів логів	Так	Так	Так		+	
3.	Наявність єдиного сховища для збору даних	Так	Ні	Так		+	
4.	Наявність візуального інтерфейсу	Так	Так	Так		+	
5.	Можливість розширення	Так	Ні	Ні			+
6.	Використання моделей машинного навчання	Так	Так	Ні			+

5.2 Технічний аудит ідеї проекту

Перед розробкою стартап-проекту було проведено технологічний аналіз проекту. Аналіз технологічної здійсненності проекту приведено в таблиці 5.3.

Таблиця 5.1 – Технологічна здійсненність ідеї проекту

№ п/п	*Ідея проекту*	*Технології її реалізації*	*Наявність технологій*	*Доступність технологій*
1.	Система збору логів	Java, Spring, Python, aiohttp	Наявні, дороблювати не потрібно.	Доступні, відкритий доступ.
2.	Центральне сховище даних	Elasticsearch, Java	Наявні, дороблювати не потрібно.	Доступні, відкритий доступ.
3.	Користувацький веб-інтерфейс	Kibana, JavaScript, jQuery, React.js	Наявні, дороблювати не потрібно.	Доступні, відкритий доступ.
4.	Модуль управління якістю надання послуг	Python, Pytorch, Django, Redis	Наявні, дороблювати не потрібно.	Доступні, відкритий доступ.
Для реалізації проекту доступні всі обрані технології: Python, Pytorch, Django, Redis, Elasticsearch, Java, Spring				

5.3 Аналіз ринкових можливостей запуску

У таблиці 5.4 наведено попередній аналіз потенційного ринку.

Таблиця 5.2. – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1.	Кількість головних гравців, од	Видимі конкуренти відсутні, наявні рішення задовольняють потребу лише частково та не є дуже поширеними.
2.	Загальний обсяг продаж, грн/ум.од	1 млрд. у. о.
3.	Динаміка ринку (якісна оцінка)	Зростає
4.	Наявність обмежень для входу (вказати характер обмежень)	Без обмежень

Продовження таблиці 5.4

5.	Специфічні вимоги до стандартизації та сертифікації	Немає
6.	Середня норма рентабельності в галузі (або по ринку), %	Дуже висока: від 50 до 80%

Наступним кроком йде аналіз потенційних клієнтів стартапу, який приведений в таблиці 5.5.

Таблиця 5.3 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*
1.	Управління якістю надання послуг критичними IT-інфраструктурами	Підприємства критичної інфраструктури, якім потрібно управляти якістю надання послуг	Залежить від галузі підприємства з критичною IT-інфраструктурою, стандарти кібербезпеки	Простота в користуванні, зрозумілий інтерфейс, вартість

Для того, щоб проаналізувати успіх стартап проекту, було визначено потенційні загрози, які приведено в таблиці 5.6.

Таблиця 5.4 – Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1.	Поява нових конкурентів	Поява конкурентів з подібним рішенням	Покращення продукту, запуск нових послуг, зміна цінової політики
2.	Швидкий технологічний прогрес	Якщо технології IT-інфраструктури будуть розвиватися дуже швидко, це може призвести до неактуальності рішення	Постійна розробка та дослідження для пошуку найкращих варіантів рішення

Аналіз потенційних можливостей приведено у таблиці 5.7.

Таблиця 5.5 – Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1.	Вільний ринок країни	Відсутність рішень на вітчизняному ринку	Швидке прийняття рішення на підприємстві
2.	Розвиток галузі	Галузь розвивається швидкими темпами та приймає високотехнологічні рішення	Можливість інтеграції та співпраці

Ступеневий аналіз конкуренції на ринку України приведений в таблиці 5.8,

Таблиця 5.6 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Тип конкуренції України – монополістичний / чистий	На ринку країни відсутні готові системи управління якістю надання послуг	Вихід на ринок, маркетингова компанія
2. За рівнем конкурентної боротьби національний та інтернаціональний	Підприємства з критичними IT-інфраструктурами є повсюди	Вихід на ринок, можливе розширення меж ринку
3. За галузевою ознакою - внутрішньогалузева	Конкуренція тільки на ринку систем управління ресурсами	Аналіз ринку
4. Конкуренція за видами товарів: - товарно-родова	Конкуренція тільки на ринку систем управління ресурсами	Аналіз ринку
5. За характером конкурентних переваг - цінова	Система дозволяє економити значні кошти через зменшення ризиків	Вихід на ринок, можливе розширення меж ринку

Наступним кроком було проведено аналіз пропозиції на ринку та визначення загальної конкурентоспроможності продукту. Аналіз приведений в таблиці 5.9.

Таблиця 5.7 – Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	*Потенційні конкуренти*	*Постачальники*	*Клієнти*	*Товари-замінники*
Складові аналізу	*Прямі конкуренти на вітчизняному ринку відсутні; На міжнародному ринку – Datadog, SignalFx*	*Вихід світових конкурентів на вітчизняний ринок*	*Постачальників немає*	*Обмежена платоспроможність*	*Товарозамінники відсутні*
Висновки:	Конкуренція на вітчизняному ринку відсутня	Можливість виходу на ринок є. Потенційні конкуренти не знають ринок України, час виходу до двох років	Постачальників нема	Можлива відмова з боку клієнта	*Товарозамінники відсутні*

Наступним кроком після аналізу конкуренції в галузі за М. Портером, проводиться більш глибокий та детальний аналіз факторів конкурентоспроможності стартап-проекту. Було виділено п’ять основних факторів, на основі яких було проведено аналіз. До таких факторів можна віднести ціну – адже ціна на продукт може напряму формувати попит, інноваційність забезпечує конкурентну перевагу, відкритість, масштабованість та простота міграції – це все технічні фактори, які впливають на конкурентоспроможність. Аналіз факторів конкурентоспроможності приведений в таблиці 5.10

Таблиця 5.8 – Обґрунтування факторів конкурентоспроможності

№ п/п	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1.	Ціна	Система дозволяє економити гроші, які пішли б на вирішення потенційних критичних проблем
2.	Відкритість	Використання відкритих стандартів дозволяє системі бути гнучкою та надає можливість розробникам розширювати функціонал за потреби
3.	Інноваційність	Використання найновіших алгоритмів та підходів дозволяє точніше передбачати стан системи та виправляти помилки, пов’язані з цим
4.	Масштабованість	За потреби система може бути автоматично масштабована для підтримки більшої кількості користувачів
5.	Простота міграції	В разі, якщо систему потрібно мігрувати на інші сервери, код системи зберігається в єдину базу і може бути швидко розгорненим на новій базі даних

Аналіз сильних та слабких сторін проекту приведено в таблиці 5.11.

Таблиця 5.9 – Порівняльний аналіз сильних та слабких сторін «системи управління якістю надання послуг критичними IT-інфраструктурами»

№ п/п	*Фактор конкурентоспроможності*	*Рейтинг товарів-конкурентів у порівнянні з … (назва підприємства)*						
		–3	*–2*	*–1*	*0*	*+1*	*+2*	*+3*
1	Ціна		+					
2	Відкритість				+			
3	Інноваційність			+				
4	Масштабованість			+				
5	Простота міграції				+			

Проведений SWOT аналіз ідеї стартап проекту наведений у таблиці 5.12.

Таблиця 5.10 – SWOT- аналіз стартап-проекту

Сильні сторони: 1. Інноваційність 2. Ціна	Слабкі сторони: 1. Досвід 2. Наявність даних
Можливості: 1. Несформована конкуренція 2. Наявність потреби ринку	Загрози: 1. Поява нових конкурентів 2. Швидкий технологічний прогрес

Було проведено аналіз альтернативних шляхів впровадження проекту на ринок для розширення можливих шляхів потрапляння проекту на ринок, який приведений в таблиці 5.13. Обидва проаналізовані шляхи мають хорошу ймовірність отримання ресурсів, а строки реалізації від одного до двох років.

Таблиця 5.11 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	*Альтернатива (орієнтовний комплекс заходів) ринкової поведінки*	*Ймовірність отримання ресурсів*	*Строки реалізації*
1	Спільні продажі	Дуже висока, у випадку знаходження великої компанії-інвестора	1 рік
2	Індивідуальні продажі	Середня	2 роки

Серед двох альтернатив перевага надається тій, для якої:

- ймовірність отримання ресурсів найбільша;
- строки реалізації найменші.

В даному випадку хорошою альтернативою можуть стати спільні продажі.

5.4 Розроблення ринкової стратегії стартапу

На ринку є попит на системи управління ресурсами критичних IT-інфраструктур, хоча такий ринок дуже обмежений В таблиці 5.14 проведено аналіз цільових груп споживачів продукту.

Таблиця 5.12 – Вибір цільових груп потенційних споживачів

№ п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовний попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*
1	Підприємства з критичною IT-інфраструктурою	Готові	Високий попит	Середня	Середня
Які цільові групи обрано: Підприємства з критичною IT-інфраструктурою					

У таблиці 5.15 було визначено базову стратегію розвитку стартапу.

Таблиця 5.13 – Визначення базової стратегії розвитку

№ п/п	*Обрана альтернатива розвитку проекту*	*Стратегія охоплення ринку*	*Ключові конкурентоспроможні позиції відповідно до обраної альтернативи*	*Базова стратегія розвитку**
1	Продаж продукту середньому та великому бізнесу	Вибірковий розподіл	Здатність протистояти прямим конкурентам. Невеликі витрати та можливість співпраці	Диференціація

У таблиці 5.16 приведено аналіз базової стратегії конкурентної поведінки.

Таблиця 5.14 – Визначення базової стратегії конкурентної поведінки

№ п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки**
1	Так	Шукати нових і забирати існуючих	Ні	Спрямованість

У таблиці 5.17 результати аналізу стратегії позиціонування продукту на ринку.

Таблиця 5.15 – Визначення стратегії позиціонування

№ п/п	*Вимоги до товару цільової аудиторії*	*Базова стратегія розвитку*	*Ключові конкурентоспроможні позиції власного стартап-проекту*	*Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)*
1	1. Ціна 2. Відкритість 3. Інноваційність 4. Масштабованість 5. Простота міграції 6. Якість	Стратегія спрямованості	- Ціна - Інноваційність - Відкритість	- Оптимізація витрат - Покращення точності передбачень - Підвищення прибутку

5.5 Розроблення маркетингової програми

Маркетингова програма повинна включати в себе ідеї щодо того, як виводити продукт на ринок, як знаходити потенційних клієнтів та збільшувати клієнтську базу та як позиціонувати новий продукт на ринку. Аналіз ключових переваг концепції потенційного продукту стартапу, який приведений у таблиці 5.18.

Таблиця 5.16 – Визначення ключових переваг концепції потенційного товару

№ п/п	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити*
1	Збільшення прибутку	Покращення якості надання послуг	Відкритість, простота, інноваційність
2	Оптимізація витрат	Зменшення витрат через критичні ситуації	Відкритість, простота, інноваційність
3	Підвищення якості надання послуг	Зменшення часу відгуку системи	Зменшення часу відгуку системи

Наступним кроком було проведено аналіз трьох рівнів моделі товару, результати якого приведені у таблиці 5.19.

Таблиця 5.17 – Опис трьох рівнів моделі товару

Рівні товару	*Сутність та складові*		
І. Товар за задумом	Продукт у вигляді системи управління якістю надання послуг критичними ІТ-інфраструктурами		
	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Кількість		1 шт
	Якість: стандарти кібербезпеки		
	Пакування: через пакет, що встановлюється		
	Марка: Duukie		
	До продажу: програмний продукт		
	Після продажу: підтримка, поширення, оновлення		
За рахунок чого потенційний товар буде захищено від копіювання: закон про захист інтелектуальної власності			

Визначення меж встановлюваної ціни приведено у таблиці 5.20.

Таблиця 5.18 – Визначення меж встановлення ціни

№ п/п	*Рівень цін на товари-замінники*	*Рівень цін на товари-аналоги*	*Рівень доходів цільової групи споживачів*	*Верхня та нижня межі встановлення ціни на товар/послугу*
1	Від 100$ за ліцензію	Від 150$ за ліцензію	Від 4000$ в місяць	Нижня – 50, верхня – 150

Аналіз формування системи збуту приведений у таблиці 5.21.

Таблиця 5.19 – Формування системи збуту

№ п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
1	Оплата підписки через веб-інтерфейс	Доступ до додаткових функцій сервісу	Багаторівневий	Селективна з використанням комбінованого підходу

Аналіз концепцій маркетингових комунікацій приведено в таблиці 5.22.

Таблиця 5.20 – Концепція маркетингових комунікацій

№ п/п	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуються цільові клієнти*	*Ключові позиції, обрані для позиціонування*	*Завдання рекламного повідомлення*	*Концепція рекламного звернення*
1	Безкоштовне налаштування системи	Прямі офіційні	Доступність, якість, швидкодія та простота	Розширення аудиторії	Наголосити на використанні інноваційних технологій та швидкодії

5.6 Висновки до розділу

В даному розділі було проведено маркетинговий аналіз стартап-проекту системи управління якістю надання послуг критичними IT-інфраструктурами. В ході аналізу було встановлено, що потенційний ринок для реалізації продукту розвивається швидкими темпами, а конкурентів, які надають схожі послуги небагато. До того ж н ринку України взагалі відсутні такі рішення, що дозволяє потенційно швидко вийти на ринок.

В ході аналізу конкурентів продукту було виявлено, що хоч потенційні конкуренти існують на світовому ринку, проте вони мають багато обмежень щодо використання в задачі управління якістю надання послуг критичними IT-інфраструктурами. На ринку України такі конкуренти відсутні.

В ході всестороннього маркетингового аналізу було встановлено сильні та слабкі сторони проекту. До сильних сторін можна віднести інноваційність, ціну та простоту користування, що забезпечує очевидну конкурентну перевагу. Отже, в підсумку, стартап можна потенційно виводити на ринок.

ВИСНОВКИ

В результаті виконання даної роботи було розроблено програмне забезпечення системи управління якістю надання послуг критичними IT-інфраструктурами. В рамках розробки програмного забезпечення було розроблено структурну схему системи управління якістю надання послуг критичними IT-інфраструктурами, схеми алгоритмів роботи модулю та інші схеми та діаграми. В структурній схемі було виокремлено чотири основні компоненти системи – це власне критична IT-інфраструктура підприємства, яка складається з серверів додатків та баз даний, які можуть дублюватися для забезпечення стабільної роботи системи.

Було розроблено структурну схему для серверу системи управління якістю надання послуг критичними IT-інфраструктурами. Було також розроблено структурну схему для сервера візуалізації даних, що складається з декількох модулів. А для візуального представлення цих даних використовується користувацький веб-інтерфейс. В якості системи для моніторингу логів було використано ELK Stack.

Для розробки компонентів системи було використано підхід по управлінню розробкою програмного забезпечення під назвою Agile. Використання такого підходу дозволило зменшити час на впровадження нових змін в продукт, що забезпечило гнучкість у впровадженні нових змін та водночас підтримувало постійний темп розробки.

Для розгортання системи на серверах підприємства було розроблено схему розгортання компонентів і було проведено розгортання основних компонентів системи – ELK Stack та модулю системи управління якістю надання послуг критичними IT-інфраструктурами.

Було розроблено алгоритм управління якістю надання послуг критичною IT-інфраструктурою підприємства. В ході виконання основного циклу програми набір даних потрапляє після попередньої обробки до моделі машинного навчання, яка намагається передбачити стан системи в майбутньому і вплинути на стан за потреби.

Було порівняно різні види сучасних архітектур моделей штучних рекурентних нейронних мереж та було обрано таку, яка найкраще відповідатиме поставленим задачам. Таким алгоритмом було обрано рекурентні нейронні мережі на основі GRU.

Як перший етап розробки програмного забезпечення підсистеми управління якістю надання послуг критичними IT-інфраструктурами, було розроблено схему класів даного програмного забезпечення.

Для оцінки якості моделі було вирішено використовувати метрики для оцінки якості класифікації – оцінка f1 та AUC ROC. Такі метрики найкраще відображають відношення правильності та точності, а також їх можна використовувати у випадку незбалансованості даних, що є актуальним для даної роботи.

Було протестовано результат роботи даного програмного забезпечення на згенерованих даних та отримано результат. На згенерованих даних результат тренування моделі був хорошим, таким чином експериментально було доведено, що використання такої системи управління ресурсами критичними IT-інфраструктурами, може допомогти передбачати критичні моменти в системі за деякий час до їх появи, що може допомогти виправляти такі ситуації ще до їх виникнення.

На основі аналізу маркетингової стратегії стартап-проекту, було визначено слабкі сторони проекту на ринку, до яких може належати обмежена цільова аудиторія. До сильних сторін проекту належить ціна продукту, швидкодія продукту, простота у використанні, наявність інноваційних рішень на основі штучного інтелекту та високий попит на рішення такого типу на вітчизняному ринку, разом з невеликою кількістю конкурентів. Все це може бути підставою до швидкої окупності проекту.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gartner ITBudget: Enterprise Comparison Tool [Електронний ресурс] // Gartner. – 2017. – Режим доступу до ресурсу: http://www.gartner.com/downloads/public/explore/metricsAndTools/ITBudget_Sample_2012.pdf.
2. 2003: The great North America blackout [Електронний ресурс] – Режим доступу до ресурсу: https://www.cbc.ca/archives/entry/2003-the-great-north-america-blackout
3. Massive power cut hits India [Електронний ресурс] – Режим доступу до ресурсу: http://news.bbc.co.uk/2/hi/south_asia/1096957.stm
4. Service Level Agreement (SLA) [Електронний ресурс] – Режим доступу до ресурсу: http://www.legalservicesboard.org.uk/Projects/pdf/service_level_agreement_guidance_final_version.pdf.
5. ITIL Maturity Model [Електронний ресурс] – Режим доступу до ресурсу: https://www.axelos.com/best-practice-solutions/itil/itil-maturity-model.
6. Amazon Web Services [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Amazon_Web_Services.
7. Amazon AWS Cloud Solutions [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/solutions/?nc1=h_ls.
8. Chowdhury N. Network virtualization: state of the art and research challenges / N. Chowdhury, R. Boutaba., 2009. – 6 с. – (IEEE Communications Magazine).
9. Burchard L. The virtual resource manager: an architecture for SLA-aware resource management / L. Burchard, M. Hovestadt. – Chicago, IL, USA: IEEE, 2004. – (IEEE International Symposium on Cluster Computing and the Grid).
10. Fazio M. Virtual Resource Management Based on Software Transactional Memory / M. Fazio, A. Puliafito. – Toulouse, France: IEEE, 2011. – (2011 First International Symposium on Network Cloud Computing and Applications).

11. Nguyen Van H. Autonomic virtual resource management for service hosting platforms / H. Nguyen Van, F. Dang Tran, J. Menaud. – Vancouver, BC, Canada: IEEE, 2009. – (2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing).

12. Sorkunlu N. Tracking System Behavior from Resource Usage Data / N. Sorkunlu, V. Chandola, A. Patra. – Honolulu, HI, USA: IEEE, 2017. – (2017 IEEE International Conference on Cluster Computing (CLUSTER)).

13. Теленик С.Ф. УПРАВЛІННЯ НАВАНТАЖЕННЯМ І РЕСУРСАМИ ЦЕНТРІВ ОБРОБЛЕННЯ ДАНИХ ПРИ ВИДІЛЕНИХ СЕРВЕРАХ [Електронний ресурс] / Теленик С.Ф., Ролік О.І., Букасов М.М. – Режим доступу до ресурсу: http://aaecs.org/telenik-sf-rolko-bukasov-mm-rimarrv-rolk-ko-upravlnnya-na-vantajennyam--resursami-centrv-obroblennya-danih-pri-vidlenih-serverah.html.

14. Ролик А.И. Модель управления перераспределением ресурсов информационно-телекоммуникационной системы при изменении значимости бизнес-процессов// Автоматика. Автоматизация. Электротехнические комплексы и системы. ХГТУ, 2007. — №2(20).— С. 73—82.

15. 10-fold Crossvalidation [Електронний ресурс] – Режим доступу до ресурсу: https://www.openml.org/a/estimation-procedures/1.

16. Кеннет Р. Автостопом по Python / Р. Кеннет, Т. Шлюссер., 2017. – 336 с.

17. Obe R. Elasticsearch: Up and Running / R. Obe, L. Hsu., 2012. – 145 с.

18. Raschka S. Python Machine Learning / Sebastian Raschka., 2015.– 454 с.